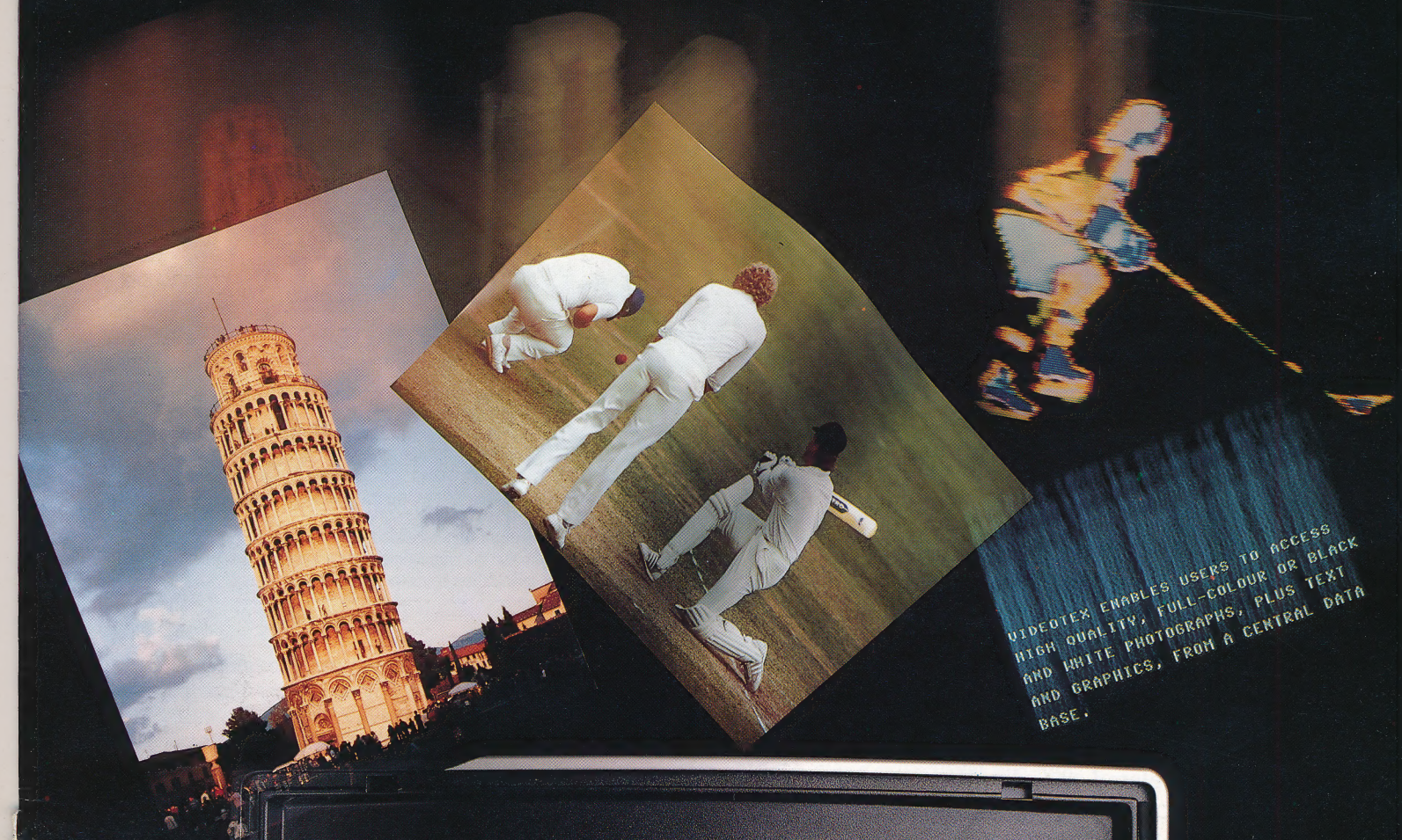
ISSN 0265-2919

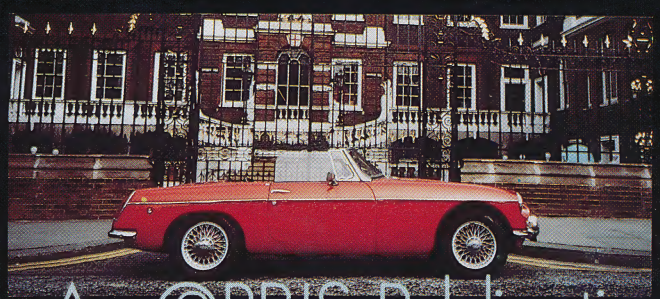
90p 88

# THE HOME COMPUTER ADVANCED COURSE

## MAKING THE MOST OF YOUR MICRO



VIDEOTEX ENABLES USERS TO ACCESS HIGH QUALITY, FULL-COLOUR OR BLACK AND WHITE PHOTOGRAPHS, PLUS TEXT AND GRAPHICS, FROM A CENTRAL DATA BASE.



An ORBIS Publication

IR£1.15 Aus $2.15 NZ $2.65 SA R2.45 Sing $4.50

# CONTENTS


APPLICATION

**VISUAL TRANSMISSION** We take a look at videotex systems, which allow the transmission of high-quality images 1741

HARDWARE

**MAKING A DISPLAY** Our series on the internal workings of a computer looks at the way a video chip controls a screen display 1750

SOFTWARE

**STANDARD BEARER** MS-DOS began life as a 'quick and dirty operating system', and this is perhaps one of the reasons why it has been so successful 1744

**WORDS WITH A MOUSE** We look at MacWrite, Apple's word processing package for the Macintosh 1752

COMPUTER SCIENCE

**STRUCTURAL SURVEY** A discussion of the control structures available in C 1755

JARGON

**TELEPRINTER TO TEXT EDITOR** A weekly glossary of computing terms 1749

PROGRAMMING PROJECTS

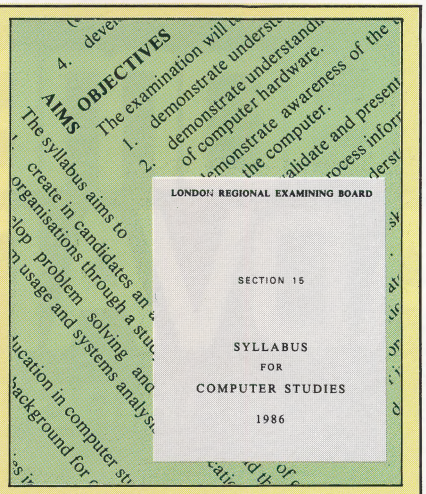
**PLAYING BY THE RULES** We develop the routines that deal with storing and evaluating each hand in our pontoon game 1746

MACHINE CODE

**READ THE INSTRUCTIONS** The instruction set of the 68000 can be divided into groups of related opcodes. We consider two of these groups — the data copying and computational instructions 1758


## Next Week

**•We conclude our brief examination of the internal electronics of a microcomputer.**
**•As computer literacy is seen to be increasingly important in the modern world, so computer science courses at school are becoming more popular. We look at how secondary schools are developing the subject.**
**•Our series on MS-DOS continues with a look at some of the commands implemented in the operating system.**




## QUIZ

1) What major advances were included in MS-DOS versions 2.0 and 3.0?
2) What is the difference between the ADDA and ADDI opcodes in the 68000 instruction set?
3) What feature distinguishes a 'dumb' from an 'intelligent' terminal?
4) What is the purpose of the MacWrite 'clipboard'?

**Answers To Last Week's Quiz**
**1)** The virtual device interface provides communication between the operating system itself and the device drivers.
**2)** PIO chips are incorporated into the memory map to enable the registers to be accessed by the processor, which makes them 'programmable'.
**3)** It is useful for an expert system to be able to explain its reasoning so that the programmers and users can check whether the decisions taken are correctly reasoned.

## Coming Up

•A series detailing the range of careers available in the computing industry.
•A look at developments expected in the computing industry in the near future.


**Editor** Stephen Cooke; **Art Editor** Claudia Zeff; **Deputy Editor** Steve Colwill; **Production Editor** Bobby Pickering; **Designer** Julian Dorr; **Staff Writer** Steve Malone; **Art Assistant** Caroline Clayton; **Sub Editor** Jon Kaye; **Contributors** Mike Curtis, Steve Colwill, Steve Malone, Nigel Cross, David Fensome, Tony Dennis, Peter Shaw; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd; Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Susan Brown; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

**HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE** – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA –** Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
**UK/EIRE** – Issue Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Issue Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Issue Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Issue Price: US/CANS1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Issue Price: SA R2.45. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Issue Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Issue Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Issue Price: AusS2.15. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Issue Price: NZS2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
**ADDRESS FOR BINDERS AND BACK ISSUES** – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
**NOTE** – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
**ADDRESS FOR SUBSCRIPTIONS** – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY CRISPIN THOMAS MONITOR COURTESY OF SANYO, 80 REGENT ST, LONDON W.1.

# VISUAL TRANSMISSION

**Videotex systems send high-quality images via telephone lines or the airwaves to computers. The enormous potential and relatively inexpensive operating costs of these systems mean that they are finding increasing favour with micro owners. We look at the principles behind videotex, and consider some successful systems.**

Most home computer users have grown accustomed to a level of computer graphics that is well below photographic quality. A higher standard of picture, however, is just one of the technical capabilities of what has come to be known as 'videotex'. Confusingly, the word videotex has acquired two related but differing meanings, depending on whether it is used in the UK or the US. The former covers visual and textual information distributed by either television transmitter or cable. In the US, the term 'videotex' is applied to a mixture of text and video transmitted solely by cable.

The drawback with any high-quality videotex system is the speed at which images appear on the screen. While a typical Prestel viewdata page generally takes under a second to fill the screen, it can take 10 seconds or longer for the screen to fill in other videotex systems. The reason lies in the amount of data that needs to be transmitted for a single screen to appear. British Telecom's Prestel and Photo Videotex systems make a good comparison.

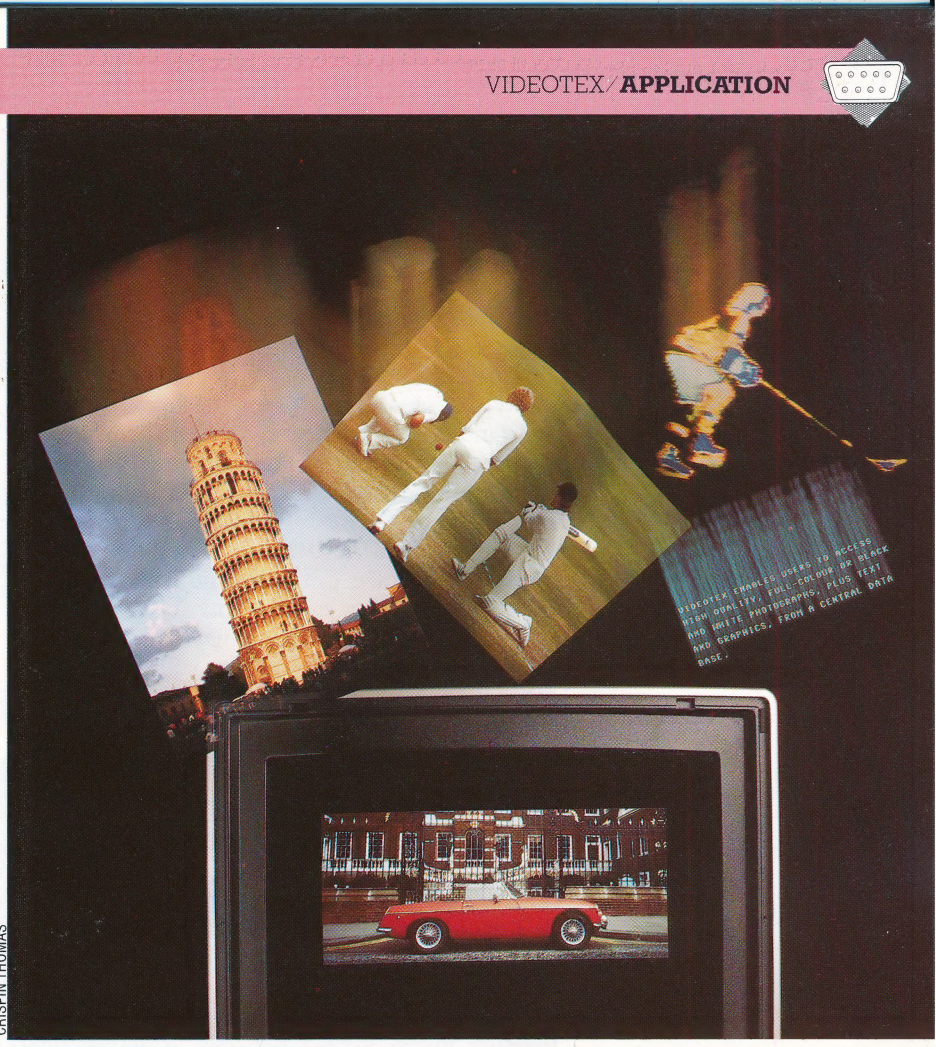
Screen definitions are quantified in pixels. A typical Prestel page will be 234 pixels by 22 lines. By contrast, a Photo Videotex screen has a definition of 270 pixels by 240 lines, but pays for the increased definition with the far higher data requirement to store such a picture. (Photo Videotex requires 128 Kbytes whereas a Prestel page needs five Kbytes at most.)

Data compression is playing an increasingly important part in lowering the data requirement. Photo Videotex, for example, is able to compress a theoretical 128 Kbyte requirement to an actual 64 Kbytes. Eventually, 'intelligent' systems will be able to sample the screen and send data only for the parts of the picture that have changed. The resulting amount of data needing to be sent will be dramatically lowered.

Videotex can be used for any application in which text needs to be mixed with colour illustrations. For example, in security systems, which are major users, videotex helps to match faces to names. With this arrangement, a guard is able to call up onto a monitor records held on a central database. Against a keyed-in name, a record will appear containing a colour photo of the individual plus a certain amount of personal information. Should the guard be uncertain that the individual matches the photo, it is a simple task to double-check and ask for a date of birth, national insurance number and so on.

Videotex is also a major advertising medium. Travel agents and estate agents already make use of viewdata systems like Prestel and Compunet (see page 963) to match buyers to vendors. Further developments to this approach might lead to monitors being placed in the shop window and set to carousel through frames held on a database. Passers-by could therefore see details of the latest offers or houses for sale. Since videotex is interactive, purchasing decisions can also be relayed back. Travel agents, for example, could make definite bookings with travel companies. If a telephone line or a private network is being used, the two firms would not even need to be in the same country.

Home shopping is available on Prestel (Club 403) from an independent service called the Armchair Grocer and via television cable. Videotex will allow users to browse through




CRISPIN THOMAS

**Video Sales**
Videotex technology enables users to transmit photographic images, text and computer graphics downline to remote terminals, thereby generating many different applications at home and in commerce. Local news, classified advertising and point of sale information can all be processed and linked to interactive systems to facilitate user feedback. This could be anything from booking holidays abroad to — in the future — viewing and commenting upon local planning proposals

shopping catalogues in the comfort of their homes. Purchases can be made either by giving a credit card number or by means of direct electronic funds transfer.

The boundaries of videotex are already being stretched to include 'video conferencing'. An American company, Datapoint, has produced a device called the Minx, which contains a video camera, a speaker telephone, a colour monitor and a much smaller reference screen. Using a local area network (Arcnet) or data line, the two parties can call up and see the person they are speaking to. The Minx has added facilities since it is also compatible with the IBM PC. The line which is carrying the voice call will simultaneously take both video and data signals. Thus it is possible for both parties to view the same data on screen during the conversation, while still seeing each other on the smaller reference screens. Files can be exchanged in the same manner. Such a system requires a two megabyte per second data link.

## THE GNOME AT HOME

Initially, viewdata services required a mainframe or minicomputer to store the data and handle callers. This situation is being increasingly eroded by microcomputers. On a small scale, the BBC Micro is being used by Soft Machinery to run a service known as The Gnome at Home. Employing a number of BBC Micros linked together by an Econet network, it allows eight simultaneous callers. The software is a modified version of Communitel, a package developed by the Notting Dale ITEC as a viewdata host for the BBC Micro. In many ways, Communitel is similar to a bulletin board, although it is compatible with Prestel.

Apricot has a product known simply as Apricot Viewdata. Costing between £10,000 and £16,000, it provides a private viewdata host system for eight to 16 simultaneous users. Using RS232 ports, the option is accessed via a set of modems or a local area network. Since it is Prestel-compatible, it could be linked to the Prestel system itself via a gateway. (Gateway is the generic term for a link between two computers in a viewdata system.)

British Telecom's Photo Videotex system also uses a microcomputer — the IBM PC — which combines colour or black and white images input from a PAL or RGB camera with standard text. The quality of picture is equal to that of a colour photograph. The system is both accessed by and run on IBM PCs, and connections are achieved via the Public Switched Telephone Network (PSTN), KiloStream or cable television.

Videotex can provide solutions to a large variety of problems. As local area networks, fibre optic, satellite and digital telephone links come into greater use, videotex terminals will become a familiar sight in industry, commerce and even the home.

## Two Sides Of The Coin

To distinguish between the two kinds of videotex generally used in the UK, the terms 'viewdata' and 'teletext' have been coined. Viewdata is based on work done by Sam Fedida at what has become British Telecom's Research Centre at Martlesham Heath. As a result of his proposals, the world's first commercially available viewdata system came into existence in 1979 in the form of Prestel.

Prestel's form of viewdata has been a great commercial success, enabling it to penetrate extremely competitive markets like the US. Prestel, however, is by no means the only viewdata service in the UK. Private viewdata services do exist; ADP and Istel (a British Leyland subsidiary) have their own telephone-accessible systems.

While viewdata relies mainly on telephone lines, teletext transmissions are 'piggybacked' onto the standard television signal. Once again the result is a mixture of text and graphics. Specially modified television sets are needed in order to receive teletext.

COURTESY OF BRITISH TELECOM

COURTESY OF MTV

## Wired For Sight

Although cable television is an ideal medium for videotex transmissions, it is a slow starter in the UK. In 1985, it was reported that only 127,000 households received cable television. This figure will be around the 300,000 mark when the young cable companies like Westminster Cable come fully on stream.

In 1984, an experiment in cable transmission of software was carried out. Called the Gamestar Project, it was a joint venture between British Telecom and Micronet 800. Users could hire a Sinclair Spectrum and a modified Prism modem to download games software. It was taken up by some cable companies including Rediffusion, but the project was dropped because of technical difficulties and the low user base then in existence. British Telecom is now concentrating on Cabletext, a spin-off from Prestel for cable companies

**Private View**
British Telecom's Photo Videotex system uses an IBM PC and provides comprehensive editing facilities, enabling images to be cropped, moved and zoomed, and combined with overlays, text and enhanced graphics. Applications include local news, advertising, point of sale promotions, personnel records and in-house systems for large companies.

The system links user terminals, editing terminals and a central computer system, enabling two-way communication and direct response. Picture quality is approximately equivalent to that of a UK domestic TV and each full-size colour image requires 64 Kbytes of storage


In the UK there are three main teletext operators: BBC's Ceefax, ITV's Oracle and Channel 4's 4-Tel. Teletext employs pages just like its viewdata counterpart, the major difference being that teletext is not truly interactive and cannot transmit responses from the user.

Teletext is sometimes referred to as 'broadcast videotex' (the term should not to be confused with 'teletex', which is an enhanced form of telex). Technical co-operation between British Telecom and the BBC/IBA has led to a degree of compatibility between the two systems. The term 'World System' has consequently been applied to anything conforming to the Prestel or Oracle/Ceefax standards.

US manufacturers have traditionally preferred a higher degree of picture resolution than that employed by Prestel. American videotex systems mix video images generated by a camera with text from a computer. North America has no equivalent of teletext. For the sake of clarity, systems using television are referred to here as teletext, while videotex will be used for any type of system that employs cable or telephone lines

## The Right Connections

The type of cable used by a videotex system can take a variety of forms. A videotex system might form part of a local area network, for example. The cable which makes the physical connection can be a twisted pair (just two wires), a co-axial cable (like a television aerial) or a multicore cable (as in the RS232 cable used for printers and modems).

Alternatively, a British Telecom line can be used as the cable. Naturally, there are options here as well. The normal telephone line used for an everyday voice call is part of the Public Switched Telephone Network. A PSTN line can link the receiving end directly to the videotex source, as in the case of British Telecom's Prestel service.

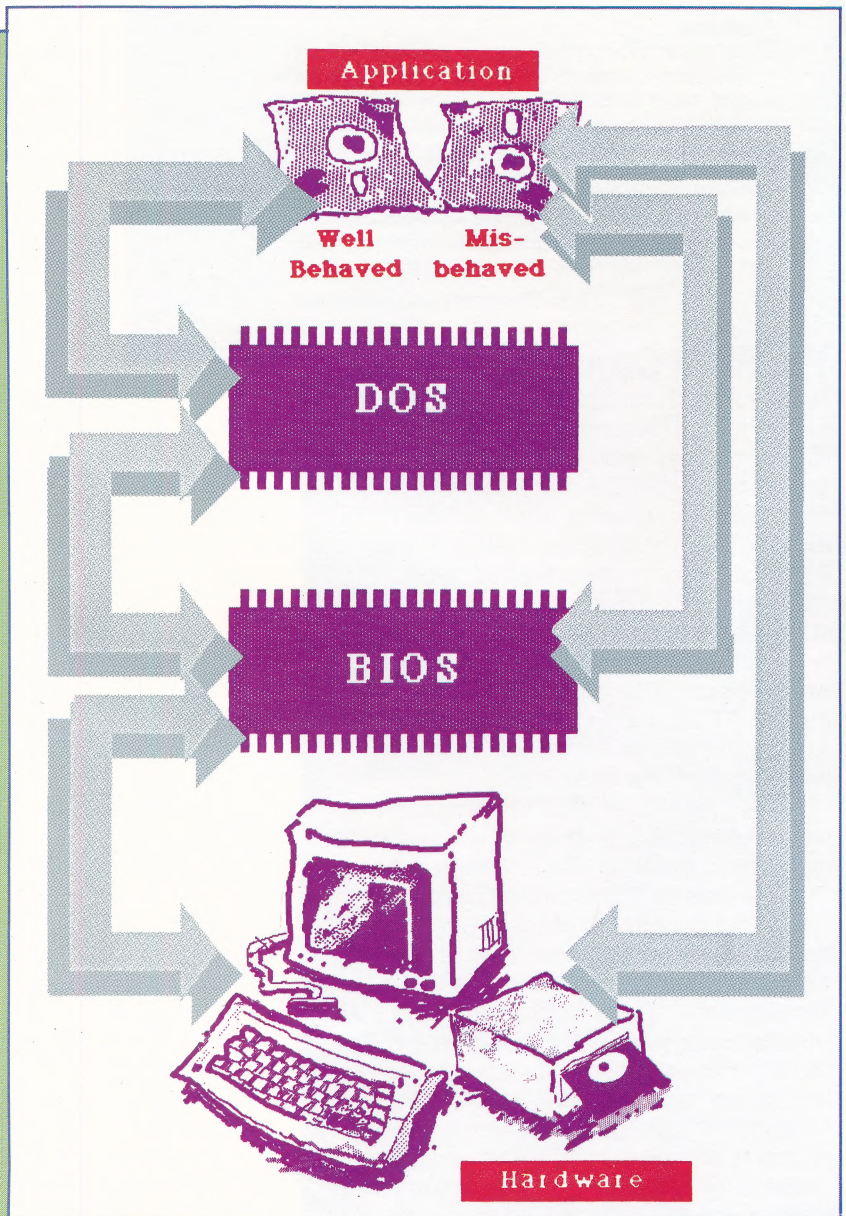
Packet SwitchStream (PSS) can be accessed by the microcomputer user via the PSTN. A local call is made using a modem to what is known as a 'node'. The node takes the data and turns it into packets (effectively self-contained parcels of data) under a protocol known as X25. The node has its own direct data links to other nodes scattered across the UK. The service even crosses national boundaries and is linked to similar services in at least 20 other countries. Naturally enough, the service is known as International Packet SwitchStream (IPSS). Packet switching can cope with data transmission speeds of up to 48 Kbits per second, and is only as good as the link to the local node — if the line is noisy, error-checking is needed and transmission speeds drop.

It is for this reason that it is better to use a private, dedicated line, which will transmit data much more cleanly. Private leased lines make up British Telecom's 'X-stream' services, such as MultiStream, KiloStream, MegaStream and even SatStream (a satellite link). British Telecom's rival, Mercury, can also provide dedicated data lines. These services are priced relative to the data transmission speed.

Such services are really stop-gap measures until the whole of the UK is converted to British Telecom's digital telephone exchange network, System X, at the end of this decade. System X's ability to switch data as well as voice calls will make videotex an easy option.

In 1985, a videotex system producing a photographic quality link required a data transmission speed of between 64 Kbits and two Mbytes per second. The economical modes for videotex applications were KiloStream, MultiStream and some local area networks

# STANDARD BEARER




CAROLINE CLAYTON

## Good Behaviour

Software that makes use of standard MS-DOS calls is termed 'well-behaved'. In this case, DOS calls the relevant BIOS routine indirectly through a 'jump table'. This 'well-behaved' route is shown in the diagram. Programs that assume knowledge of absolute addresses in the BIOS, or bypass MS-DOS completely by 'talking' directly to the underlying hardware, are considered 'misbehaved'. In the IBM PC, in particular, the BIOS is in a proprietary ROM chip and the jump table starts at address 400H. This is located 512 bytes (two hexadecimal pages) below the standard MS-DOS table base at 600H. A program that calls the BIOS or hardware directly, therefore, will only run on an IBM machine or 100 per cent compatible clone

**The introduction of the portable operating system ushered in a new era for microcomputers. Following our series on the eight-bit industry standard CP/M from Digital Research (see page 1264), we now look at Microsoft's MS-DOS, which has emerged as the 16-bit standard, and the basis for the much copied IBM PC-DOS.**

In the initial days of eight-bit microcomputers, the industry was dominated by machines using the Zilog Z80 processor and CP/M. This disk operating system was originally written for the Intel 8080 chip by Gary Kildall, who subsequently founded Digital Research.

By the time IBM decided to enter the micro arena, eight-bit systems were nearing the end of their useful life — yet no definitive combination of CPU and operating system had emerged as a 16-bit replacement for the Z80 and CP/M. IBM therefore had the option of introducing a completely new hardware and software configuration for the 16-bit market. They chose not to do so for several reasons:

1. Despite dominating the mainframe world, IBM had no experience of the radically different micro computer market.
2. The success of DEC (Digital Equipment Corporation) in the minicomputer field had shown IBM that users were no longer prepared automatically to buy IBM's products without serious assessment of their value.
3. The length of time required to develop an adequate software base for any new system would have been unacceptable in the fast-moving micro world unless IBM gained the support of third party software houses.
4. The development costs of both software and hardware would have meant unacceptably high end-user prices.

The decision was consequently taken to adopt the 'best' of what was currently available (meaning whatever was dominant in the marketplace), and adapt it where necessary for 16-bit systems.

This approach was hardly innovative, and the resulting machine, the IBM PC, is conservative in design, slow in execution speed and breaks no new ground, especially in comparison with its contemporaries, such as the Apple Lisa. Even the choice of Intel's 8088 processor and a commitment to future developments centred upon the 8086/186/286/386 series of chips (known generally as the 8086 family) was a pragmatic decision. Although the 8088 had an internal 16-bit architecture, the external data bus was only eight bits wide. This entails each 16-bit 'word' being fetched in two operations, thereby slowing down the processing speed, but enabling the use of existing well-proven (and cheap) eight-bit peripheral chips.

For an operating system and applications software, IBM approached Microsoft, a company that had become a major name in microcomputing primarily because of its popular dialect of BASIC. At the time, Microsoft was in contact with a firm called Seattle Computer Products, which was working on Intel-based 16-bit systems. In the absence of a CP/M for the 8086 family, a programmer called Tim Patterson had written a machine code DOS (called SCP-DOS) which he'd modelled closely on CP/M. His system was also known as QDOS (not to be confused with Sinclair's OS for the QL), and this stood for 'quick and dirty operating system'. That description fitted it perfectly — Patterson's OS was serviceable, but not entirely developed and

debugged. Microsoft subsequently bought the rights to the system and licensed it for use by original equipment manufacturers (OEMs).

## PC-DOS

IBM produced its own version (version 1) of MS-DOS for the IBM PC called PC-DOS, and has since kept up with the major developments in subsequent MS-DOS versions — all subsequent references in this series to an MS-DOS version can be applied equally to PC-DOS. But let's first look at some of the differences between the two. As far as the user is concerned, there are essentially none. Some internal details differ, and some trivial details such as the numbering of disk drives vary, but that's all — in theory, at least. Unfortunately, however, many applications programs have been written for the IBM PC which address the hardware directly, instead of using the operating system calls provided. This is the main reason why compatibility problems arise (it is not a problem inherent in the different versions of MS-DOS and PC-DOS). Programs that call DOS correctly are described as being 'well-behaved', and such programs should run unmodified on any machine with either MS-DOS or PC-DOS of the same *or later* versions. Naturally, applications that make use of recent enhancements to DOS in, say, version 3.1, can not be expected to operate successfully when running on an earlier version that does not support those particular features.

The first version of MS-DOS was essentially a version of CP/M-80 recoded for the Intel 16-bit family. If you refer back to our CP/M series (beginning on page 1264) you will recognise many of the commands and notice the similarities of internal structure. Tim Patterson did change several details, however, and improved the user-friendliness of many commands.

When Microsoft started working on enhancements, it had in mind the future use of Unix (the Bell Labs multi-user operating system) and, in particular, the Microsoft version, Xenix. A degree of compatibility between the two was aimed for, with common system calls being implemented. But by far the most important advance in DOS 2 from the user's point of view was the introduction of I/O 'redirection', 'pipelines' and 'hierarchical directories', all of which were ideas borrowed from Unix. We'll have a close look at the use of these powerful extensions to DOS in future instalments of this MS-DOS series; for the moment, though, we can consider DOS 2 to be 'DOS 1 plus a little Unix'.

## DOS 3

The most recent version of MS-DOS — DOS 3 — incorporates all the facilities of DOS 2 with the addition of multi-user support. Of the two main subsidiary versions — 3.0 and 3.1 — OEMs providing networking systems, such as Apricot and Research Machines Limited, are using version 3.1 with Microsoft Networks built in.

This can make the operating system unnecessarily large, however, so for other environments DOS 3.0 is available, stripped of 3.1's networking capability, but retaining facilities for file sharing and record locking. The soft solution provides a useful and effective answer to the problems of running applications that need to allow simultaneous access to common databases.

The latest Intel processor (80286), used in the IBM PC-AT, has both a memory addressing capability of three Mbytes and, built in to the hardware, the ability to lock regions of files. DOS 3.0 allows file sharing on the 8086 and 80186 by means of extra system calls that can temporarily deny access to a region. This prevents 'access collision', but at the expense of processing speed (extra time being taken by the system calls). Other improvements have been made, and although one or two remnants of the Seattle 'quick and dirty' DOS are still to be found, MS-DOS can be seen as a mature product with a future.

Among the most important enhancements currently available is Microsoft Windows. Like DR's GEM, this provides a WIMP-like environment together with facilities for multi-tasking and 'cut and paste' operations between applications. It is being implemented as an extension to the basic MS-DOS/PC-DOS operating system, and is designed to herald a new era in the ease of use of otherwise conventional computers. In the next instalment, we will examine the common nucleus of conventional MS-DOS commands and their use.

### Cloning Around

The main cause of early problems of compatibility was that IBM moved a 'jump table' down two (hexadecimal) pages in memory and placed it in a ROM at address 400H (the MS-DOS table starts at 600H). This part of the BIOS (basic input/output system) is the subject of IBM copyright, and poses problems for other manufacturers who wish to make 100 per cent IBM-compatible machines ('clones'). However, if all access to the underlying hardware is made via MS-DOS by means of the correct documented system calls, there is no problem in producing software that can run on any MS-DOS computer.

There is a small price to be paid, however. The system calls themselves add a slight overhead to the execution speed and, perhaps more significantly, any advantages accruing from a more advanced hardware specification may not be able to be realised. For these reasons, several software houses have written hardware — or firmware — dependent programs that need a hardware environment very closely matched to that of the IBM PC. It is also not unknown for software vendors deliberately to make their products non-portable. Digital Research, for example, alters a few bytes of code to make it necessary to buy different versions (of GEM applications, for instance) for each machine — even the most compatible of clones

PHOTOGRAPHY BY CHRIS STEVENS

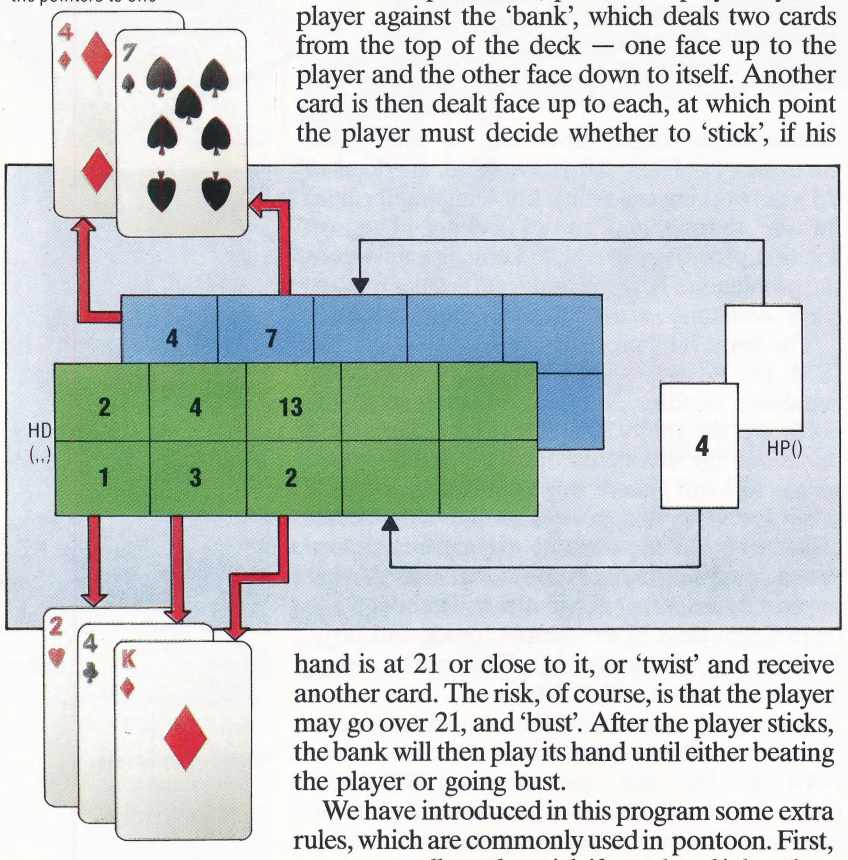
**Displaying Dominance**
The dominance of the IBM PC and its adoption of MS-DOS (under the label PC-DOS) has assured MS-DOS first place among 16-bit business operating systems. In fact, the system started life as a rough-and-ready customisation of CP/M by an ex-Digital Research employee. It was subsequently bought by Miscrosoft, who licensed it to IBM

# PLAYING BY THE RULES

**In our pontoon programming project so far, we have put together the routines that handle the card display, shuffling and dealing. We now look at how a player's hand is stored, and design a routine that will evaluate the hand at any stage of the game.**

Before looking at the programming for this instalment, let's briefly outline the rules our program will play by — there are as many versions of the game as there are names. All, however, have several fundamental aspects in common, the most relevant of which is that the player tries to get as close to 21 without going over. The cards in our program carry the following values: numbered cards score according to their face values, picture cards (jack, queen and king) score 10, and aces either one or 11, at the player's discretion.

In its simplest form, pontoon is played by one player against the 'bank', which deals two cards from the top of the deck — one face up to the player and the other face down to itself. Another card is then dealt face up to each, at which point the player must decide whether to 'stick', if his hand is at 21 or close to it, or 'twist' and receive another card. The risk, of course, is that the player may go over 21, and 'bust'. After the player sticks, the bank will then play its hand until either beating the player or going bust.




**Giving Us A Hand**
A three-dimensional array, HD(,,), is used to hold the player's and banker's hands as cards are dealt from the deck. A second array, HP(), holds pointers to the next free space in each section of the hand array. At the end of each game the program does not need to clear the array; it merely has to reset the pointers to one

We have introduced in this program some extra rules, which are commonly used in pontoon. First, you are not allowed to stick if your hand is less than 17. Secondly, if your score is 12, 13 or 14 after being dealt the first two cards, you have the option of having your cards 'burned', and having another two dealt. In the event of a tie between the bank and yourself, the bank always wins.

Although we've looked at dealing cards from the pack and displaying them (see page 1729), we didn't show you how the dealt cards could be 'remembered' by the program. To hold the player's and banker's hands, we'll set up a three-dimensional array HD(,,). Since neither side will ever require more than five cards, this array is dimensioned in line 550 to be two elements (for the two players) by five (for the cards) by two (to hold the card numbers and suits separately). A second array, HP(), is used to hold a pointer to the next free space in the hand array for each player.

We need, at this stage, to add line 1325 to the deal routine developed previously so that the card number, CN, and the suit number, SU, are added into the hand array and the hand pointer incremented. Note that if PL=1 on entering this routine, the card dealt is entered into the player's half of the hand array; if PL=2, the card details are entered into the banker's half.

The first four cards in the game can now simply be dealt by setting PL to 1 or 2 and calling the deal routine. Because the first card to be dealt to the banker must be face down, the deal routine is written in such a way that if a flag (FL) is set to 1, the card back display routine is called. The details of the hidden card are entered into the hand array.

## EVALUATING A HAND

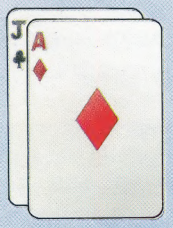
The subroutine at line 800 is designed as a general-purpose evaluation routine that performs two functions. First, it calculates the total value held in the hand being examined, and secondly it sets variable EF to a value between 1 and 5, corresponding to the five possible states that a hand can be in. Working out the state of a hand is relatively straightforward, but what's more difficult is to calculate the total value of the hand. Although it seems at first sight that all we need to do is total the card numbers held in the hand array, picture cards all score 10 and aces score low or high. This first problem can be easily solved by testing the card number — if it's bigger than 10, then 10 will be added to the score total.

The aces problem, however, needs more thought since a hand can contain up to four aces, creating up to 16 different score permutations to deal with. Most of these permutations, in fact, aren't worthy of consideration since they immediately bust the hand. We can therefore follow a simple rule for dealing with aces: the first ace in the hand can score one or 11, but any subsequent aces count as low or the hand will bust.

When it comes to coding this rule, the simplest way to handle it is to have two score options: TT(PL,1), in which the first ace is counted low, and TT(PL,2), in which the first ace counts high. We use a single loop to add up the hand values, treating any aces encountered as low, and count up the number of aces. On exiting the loop, we test the aces count and if not zero we add 10 to TT(PL,2). The remainder of the evaluation routine then tests the totals in TT(PL,1) and TT(PL,2) or the hand array directly to determine which state the hand is in and sets the evaluation state flag, EF, accordingly.

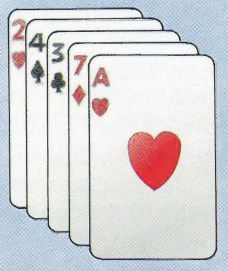
If you are dealt two cards that total 12, 13 or 14, the option is open to burn them and be dealt another two. This routine calls the evaluation routine discussed earlier and uses the totals in TT(PL,1) and TT(PL,2) to determine whether you may burn. If you choose to do so, the left half of the screen display is erased, your hand pointer, HP(PL), is reset and you are dealt two new cards.

## Scoring Schemes

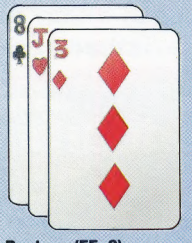


A hand in pontoon can be in five different states during a game. The following list shows the different states recognised by our pontoon program in hierarchical order. The evaluation routine sets a variable, EF, according to which state a hand is in

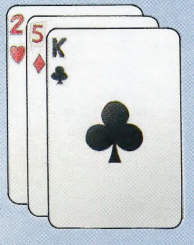
**Royal Pontoon (EF=2)**
A score of 21 from an ace and a picture card. An ace and a 10 does not count as a royal pontoon



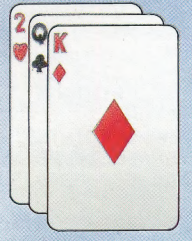
**Five Card Trick (EF=5)**
A score of 21 or less with five cards



**Pontoon (EF=3)**
A score of exactly 21



**Less than 21 (EF=1)**
A score of less than 21. This may be the state of a hand at the end of a turn or at some intermediate stage



**Bust (EF=4)**
A score of more than 21

KEVIN JONES

## Hand Storage And Evaluation Routines

### Sinclair Spectrum

```
  60>LET FL=0:LET PL=1:GO SUB 1300:REM D
EAL CARD TO PUNTER
  70 LET FL=1: LET PL=2: GO SUB 1300: RE
M DEAL CARD TO BANK
  85 LET FL=0: LET PL=1: GO SUB 1300: RE
M DEAL CARD TO PUNTER
  90 LET FL=0: LET PL=2: GO SUB 1300: RE
M DEAL CARD TO BANK
  95 REM **** PUNTER'S TURN ****
 100 LET PL=1
 102 GO SUB 2300: REM BURN OPTION
```

**Dimension Hand Arrays**

```
 550>DIM D(2,5,2):DIM P(2):REM PUNTER'S
AND BANKER'S HANDS
 555 DIM T(2,2): REM SCORE TOTALS
 670>REM **** ERASE CARD DISPLAY ****
 675 LET X(PL)=EP: LET Y(PL)=0
 680 PRINT AT 0,0;: FOR I=1 TO 12: PRINT
 AT I,EP;S$( TO 7): NEXT I
 700 REM **** PREPARE FOR INPUT ****
 710 LET TX=0: LET TY=17: GO SUB 900: PR
INT S$( TO 31)
 720 LET TX=0: LET TY=17: GO SUB 900: RE
TURN
```

**Hand Evaluation Routine**

```
 800 REM **** EVALUATE HAND ****
 810 LET AV=1: FOR J=1 TO 2
 812 LET T(PL,J)=0
 815 FOR I=1 TO P(PL)-1
 820 IF D(PL,I,1)=1 THEN  LET T(PL,J)=T(
PL,J)+AV-1
 825 IF D(PL,I,1)>10 THEN  LET T(PL,J)=T
(PL,J)+10-D(PL,I,1)
 830 LET T(PL,J)=T(PL,J)+D(PL,I,1)
 840 NEXT I: LET AV=11: NEXT J
 852 IF (T(PL,1)<=21 OR T(PL,2)<=21) AND
 P(PL)>5 THEN  LET EF=5: RETURN
 854 IF D(PL,1,1)=1 AND D(PL,2,1)>10 THE
N  LET EF=2: RETURN
 855 IF D(PL,2,1)=1 AND D(PL,1,1)>10 THE
N  LET EF=2: RETURN
 856 IF T(PL,1)=21 OR T(PL,2)=21 THEN  L
ET EF=3: RETURN
 858 IF T(PL,1)<21 OR T(PL,2)<21 THEN  L
ET EF=1: RETURN
 860 IF T(PL,1)>21 AND T(PL,2)>21 THEN
LET EF=4: RETURN
2300>REM **** BURN OPTION ****
2305 GO SUB 800: REM EVALUATE
2310 IF T(PL,1)<>T(PL,2) OR T(PL,1)<12 O
R T(PL,1)>14 THEN  RETURN
2340 GO SUB 700: PRINT "BURN (Y/N) ";
2345 LET A$=INKEY$: IF A$="" THEN  GO TO
 2345
2347 IF A$<>CHR$ 13 THEN  PRINT A$
2350 IF A$<>"Y" THEN  RETURN
2360 LET P(1)=1: REM RESET HAND POINTER
2370 LET EP=0: GO SUB 670: REM ERASE CAR
DS
2380 LET FL=0: LET PL=1: GO SUB 1300: GO
 SUB 800: REM DEAL CARD TO PUNTER
2390 LET FL=0: LET PL=1: GO SUB 1300: GO
 SUB 800: REM DEAL CARD TO PUNTER
2400 GO TO 2300: REM BURN AGAIN?
```

### BBC Micro

```
  60 FL=0:PL=1:GOSUB 1300
  70 FL=1:PL=2:GOSUB 1300
  85 FL=0:PL=1:GOSUB 1300
  90 FL=0:PL=2:GOSUB 1300
  95 REM
 100 PL=1
 102 GOSUB 2300
```

**Dimension Hand Arrays**

```
 550 DIM HD(2,5,2),HP(2)
 555 DIM TT(2,2)
 670 REM ****  ERASE CARD DISPLAY ****
 675 X(PL)=EP:Y(PL)=0
 680 PRINT TAB(0,0);:FOR I=1 TO 12:PRIN
 T TAB(EP,I);LEFT$(SP$,19):NEXT I:RETURN
```

```
  700 REM
  710 COLOUR 1:TX=0:TY=23:GOSUB 900:PRIN
T SP$
  720 TX=0:TY=23:GOSUB 900:RETURN
```

**Hand Evaluation Routine**

```
  800 REM
  810 AC=0:AV=0:TT(PL,1)=0
  815 FOR I=1 TO HP(PL)-1
  820 IF HD(PL,I,1)=1 THEN AC=AC+1
  825 IF HD(PL,I,1)>10 THEN TT(PL,1)=TT(
PL,1)+10-HD(PL,I,1)
  830 TT(PL,1)=TT(PL,1)+HD(PL,I,1)
  840 NEXT I
  845 IF AC>0 THEN AV=10
  850 TT(PL,2)=TT(PL,1)+AV
  852 IF (TT(PL,1)<=21 OR TT(PL,2)<=21)
AND HP(PL)>5 THEN EF=5:RETURN
  854 IF HD(PL,1,1)=1 AND HD(PL,2,1)>10
THEN EF=2:RETURN
  855 IF HD(PL,2,1)=1 AND HD(PL,1,1)>10
THEN EF=2:RETURN
  856 IF TT(PL,1)=21 OR TT(PL,2)=21 THEN
 EF=3:RETURN
  858 IF TT(PL,1)<21 OR TT(PL,2)<21 THEN
 EF=1:RETURN
  860 IF TT(PL,1)>21 AND TT(PL,2)>21 THE
N EF=4:RETURN
 2300 REM
 2305 GOSUB 800
 2310 IF TT(PL,1)<>TT(PL,2) OR TT(PL,1)<
12 OR TT(PL,1)>14 THEN RETURN
 2340 GOSUB 700:PRINT"BURN (Y/N) ";
 2345 AN$=GET$
 2347 IF AN$<>CHR$(13) THEN PRINT AN$
 2350 IF AN$<>"Y" THEN RETURN
 2360 HP(1)=1
 2370 EP=0:GOSUB 670
 2380 FL=0:PL=1:GOSUB 1300:GOSUB 800
 2390 FL=0:PL=1:GOSUB 1300:GOSUB 800
 2400 GOTO 2300
```

## Amstrad CPC Range

```
  60 fl=0:pl=1:GOSUB 1300:REM deal card to
   punter
  70 fl=1:pl=2:GOSUB 1300:REM deal card to
   bank
  85 fl=0:pl=1:GOSUB 1300:REM deal card to
   punter
  90 fl=0:pl=2:GOSUB 1300:REM deal card to
   bank
  95 REM **** punter's turn ****
 100 pl=1
 102 GOSUB 2300:REM burn option
```

**Dimension Hand Arrays**

```
 550 DIM hd(2,5,2),hp(2):REM punter and b
anker hands
 555 DIM tt(2,2):REM score totals
 670 REM **** erase card ****
 675 x(pl)=ep:y(pl)=0:tx=ep:ty=0
 680 FOR i=1 TO 12:GOSUB 900:PRINT SPACE$
(19)
 690 ty=ty+1:NEXT i:RETURN
 700 REM **** prepare for input ***
 710 PEN white:tx=0:ty=23:GOSUB 900:PRINT
  SPACE$(39)
 720 tx=0:ty=23:GOSUB 900:RETURN
```

**Hand Evaluation Routine**

```
 800 REM **** evaluate hand ****
 810 ac=0:av=0:tt(pl,1)=0
 815 FOR i=1 TO hp(pl)-1
 820 IF hd(pl,i,1)=1 THEN ac=ac+1
 825 IF hd(pl,i,1)>10 THEN tt(pl,1)=tt(pl
,1)+10-hd(pl,i,1)
 830 tt(pl,1)=tt(pl,1)+hd(pl,i,1)
 840 NEXT i
 845 IF ac>0 THEN av=10
 850 tt(pl,2)=tt(pl,1)+av
 852 IF (tt(pl,1)<=21 OR tt(pl,2)<=21) AN
D hp(pl)>5 THEN ef=5:RETURN
 854 IF hd(pl,1,1)=1 AND hd(pl,2,1)>10 TH
EN ef=2:RETURN
 855 IF hd(pl,2,1)=1 AND hd(pl,1,1)>10 TH
EN ef=2:RETURN
 856 IF tt(pl,1)=21 OR tt(pl,2)=21 THEN e
f=3:RETURN
 858 IF tt(pl,1)<21 OR tt(pl,2)<21 THEN e
f=1:RETURN
 860 IF tt(pl,1)>21 AND tt(pl,2)>21 THEN
ef=4:RETURN
2300 REM **** burn option ****
2305 GOSUB 800:REM evaluate
2310 IF tt(pl,1)<>tt(pl,2) OR tt(pl,1)<1
2 OR tt(pl,2)>14 THEN RETURN
2340 GOSUB 700:PRINT "Burn (y/n)";
2345 an$="":WHILE an$="":an$=INKEY$:WEND
2347 IF an$<>CHR$(13) THEN PRINT an$
2350 IF an$<>"y" THEN RETURN
2360 hp(pl)=1:REM reset hand pointer
2370 ep=0:GOSUB 670:REM erase cards
2380 fl=0:pl=1:GOSUB 1300:GOSUB 800:REM
deal card to punter
2390 fl=0:pl=1:GOSUB 1300:GOSUB 800:REM
deal card to punter
2400 GOTO 2300:REM burn again ?
```

## Commodore 64

```
  60 FL=0:PL=1:GOSUB 1300:REM DEAL CARD TO
   PUNTER
  70 FL=1:PL=2:GOSUB 1300:REM DEAL CARD TO
   BANK
  85 FL=0:PL=1:GOSUB 1300:REM DEAL CARD TO
   PUNTER
  90 FL=0:PL=2:GOSUB 1300:REM DEAL CARD TO
   BANK
  95 REM **** PUNTER'S TURN ****
 100 PL=1
 102 GOSUB 2300:REM BURN OPTION
 120 GOSUB 2600:REM TWIST ETC
 550 DIM HD(2,5,2),HP(2):REM PUNTER'S AND
  BANKER'S HANDS
 555 DIM TT(2,2):REM SCORE TOTALS
```

**Dimension Hand Arrays**

```
 670 REM **** ERASE CARD DISPLAY ****
 675 X(PL)=EP:Y(PL)=0
 680 PRINT CHR$(19);:FORI=1 TO 12:PRINTTA
B(EP);LEFT$(SP$,19):NEXT I:RETURN
 700 REM **** PREPARE FOR INPUT ****
 710 PRINT CHR$(5);:TX=0:TY=23:GOSUB900:P
RINTSP$
 720 TX=0:TY=23:GOSUB900:RETURN
```

**Hand Evaluation Routine**

```
 800 REM **** EVALUATE HAND ****
 810 AC=0:AV=0:TT(PL,1)=0
 815 FOR I=1 TO HP(PL)-1
 820 IF HD(PL,I,1)=1 THEN AC=AC+1
 825 IF HD(PL,I,1)>10 THEN TT(PL,1)=TT(PL
,1)+10-HD(PL,I,1)
 830 TT(PL,1)=TT(PL,1)+HD(PL,I,1)
 840 NEXT I
 845 IF AC>0 THEN AV=10
 850 TT(PL,2)=TT(PL,1)+AV
 852 IF (TT(PL,1)<=21 OR TT(PL,2)<=21)AND
  HP(PL)>5 THEN EF=5:RETURN
 854 IF HD(PL,1,1)=1 AND HD(PL,2,1)>10 TH
EN EF=2:RETURN
 855 IF HD(PL,2,1)=1 AND HD(PL,1,1)>10 TH
EN EF=2:RETURN
 856 IF TT(PL,1)=21 OR TT(PL,2)=21 THEN E
F=3:RETURN
 858 IF TT(PL,1)<21 OR TT(PL,2)<21 THEN E
F=1:RETURN
 860 IF TT(PL,1)>21 AND TT(PL,2)>21 THEN
EF=4:RETURN
2300 REM **** BURN OPTION ****
2305 GOSUB800:REM EVALUATE
2310 IF TT(PL,1)<>TT(PL,2) OR TT(PL,1)<1
2 OR TT(PL,1)>14 THEN RETURN
2340 GOSUB700:PRINT"BURN (Y/N) ";
2345 GET AN$:IF AN$=""THEN 2345
2347 IF AN$<>CHR$(13)THEN PRINTAN$
2350 IF AN$<>"Y" THEN RETURN
2360 HP(1)=1:REM RESET HAND POINTER
2370 EP=0:GOSUB670:REM ERASE CARDS
2380 FL=0:PL=1:GOSUB 1300:GOSUB800:REM D
EAL CARD TO PUNTER
2390 FL=0:PL=1:GOSUB 1300:GOSUB800:REM D
EAL CARD TO PUNTER
2400 GOTO 2300:REM BURN AGAIN?
```

## TELEPRINTER

A *teleprinter* is a device capable of transmitting or receiving messages from other devices. The equipment consists of a typewriter keyboard for inputting messages and a modem for transmitting or receiving the signals through the telephone network. The system will normally include a printer to produce hard copy of both input information and arriving messages.

Although they are now being replaced by more modern systems, most teleprinters still run on the paper tape system (see page 1228), whereby messages will be typed in and encoded onto the tape. This will then be fed into a device that will convert the perforations into electrical signals that can be sent down the telephone line. More modern systems display the message on a VDU screen. In this case, the message will be encoded automatically and sent by a computer once the user is satisfied with it.

## TELETEXT

Now being fitted to an increasing number of television sets, *teletext* is a broadcasting system that transmits 'pages' of text and character graphics. The text is mostly information such as news and weather, although most teletext services also offer 'fun' pages, such as children's puzzles and local goings on. The system works by broadcasting the pages of the teletext simultaneously over an unused channel from television transmitters. Specially adapted television sets are able to pick up the broadcasts and the viewer can select pages by way of a 'channel converter'.


COURTESY OF PRESTEL

**Bankable Assets**
Although the use of public information systems such as the Prestel viewdata service is becoming more popular, interactive systems have yet to become widely available. These are facilities, anticipated at the beginning of the home micro revolution, which enable users to shop, bank and book holidays and flights without leaving home. The reason for the delay in developing such systems is that the cost of installation compares unfavourably with the estimated return on investment

## TERMINAL

A *terminal* is simply a keyboard and display unit through which a user can communicate with a computer. This used to be done from a teleprinter keyboard, but is almost entirely done now with VDU systems.

There are two types of terminal, although the actual dividing line between them is not pronounced. A 'dumb terminal' has no processing power of its own but merely transmits signals to the computer, where all the processing is performed. 'Intelligent terminals', on the other hand, are basically microcomputers connected to a mainframe or minicomputer. They are capable of performing some data processing and storage, although large number crunching applications and database access is left to the larger machine.

The advantage of intelligent terminals over their 'dumb' cousins is that they take up less (expensive) time on the mainframe, keeping it free for more cost-efficient tasks.

## TESTING

The process of examining a system to ensure that it is behaving as intended is known as *testing*. When a system is completed, a number of test runs will be made to ensure that the answers given are correct. Test data will also be input for the programs to work on. This test data will often contain deliberate errors in order to ensure that the error-handling routines are working correctly. Structured programs are therefore much easier to debug, since errors in the coding can easily be located within a particular module of the program.

If the program is structured, each of the modules can be tested independently. This is known, naturally enough, as 'module testing'. When each is working satisfactorily, the modules can be tested to see if they work together. This is known as 'integration testing'.

## TEXT EDITOR

A *text editor* is a program, or number of programs, which enable the user to alter or change text within a document or program. Although text editors are an essential part of any computer system, they have many features in common with word processing packages (see our series beginning on page 1695). In order to perform its job properly, a text editor has to be able to place a cursor at any position in the text while having the capacity to insert and delete characters where necessary.

There are two major types of text editor commonly in use on home micros (though many computers have features that overlap between the different types). A 'screen editor' — the Commodore 64 offers the most common example on a home micro — allows the user to move the cursor over the entire screen and edit accordingly. On the other hand, a 'line editor', such as that seen on the Sinclair Spectrum, allows only a single line to be edited, which has to be 'pulled down' from the main text.

Early operating systems like CP/M had small text editors built into them to allow the user to write short notes and memos. With the advent of sophisticated word processing packages, this function has been superceded and the text editor is now mostly used for editing programs.

# MAKING A DISPLAY

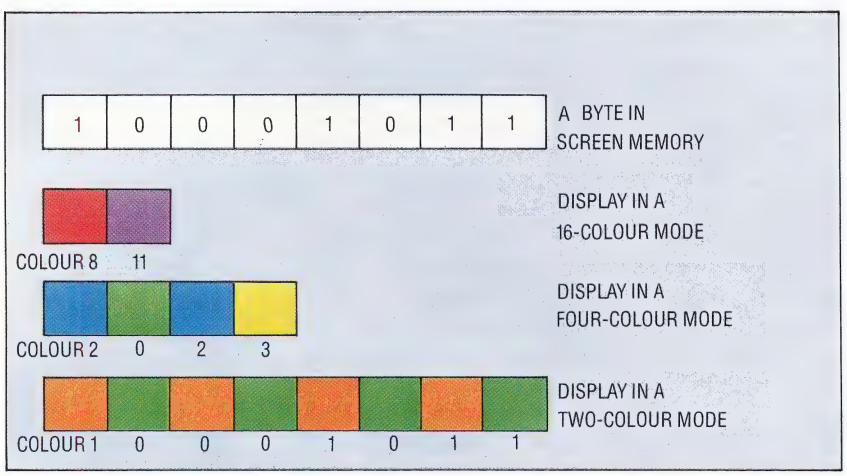
**We continue our short tour around the major components of a home computer system with a look at two different types of video display chip. We'll see how binary patterns in memory are converted into pictures on screen.**

A major selling point of any home computer is the quality and diversity of its video display. How many colours can be displayed? What is the maximum number of characters that can be displayed on a line? Does the computer have sprites? Most of these questions have their answer in the type of video chip used in the machine.

A video chip (sometimes called a CRTC — cathode-ray tube controller) has a basic function to perform — it takes display data, held as bytes in memory, and converts it to shapes and colours on the screen. Video chips differ from each other in the way that the data is converted into picture information.

Because video chips need to have close links with the memory from which they take their display information, dynamic RAM refresh circuitry (see page 1710) is normally built into the chip. As with the PIO chips, discussed in the previous instalment (see page 1732), video chips have programmable internal registers that control their functions. Normally, the direct programming of the chip is done by the computer's operating system, but special video effects can be achieved by programming the video chip yourself.

**Colour By Numbers**
Video controllers like those used in the Amstrad CPC range and the BBC Micro encode colour and pixel on/off information together in memory. On both these computers a number of screen modes are available that interpret the binary patterns held in memory in different ways. In a 16-colour mode, four bits are needed to encode each pixel's colour; thus only two pixels can be represented in each byte. In four-and two-colour modes, only two and one bits, respectively, are needed to colour code a pixel. In these modes a byte can represent four or eight pixels




The first type of video chip that we'll look at holds all its display and colour data together in memory, making the task of conversion to a picture much easier. This is the type of display used by the BBC Micro, Acorn Electron and the Amstrad CPC range. Such chips normally support a number of different display modes in which character size and the number of available colours vary. The first diagram shows how a single byte in memory will be interpreted in different screen modes. Although the video chip's work is simplified by this type of arrangement, and it is easy to mix text and high-resolution graphics, the drawback is the large amount of memory required to hold a screenful of information. A 160 by 256 pixel display with 16 colours, for example, requires 20 Kbytes of memory.

When the cost of memory was at a premium, such display methods were not feasible and an alternative display method was used, which is still present in the Commodore 64. Rather than holding every pixel of display and colour information explicitly in memory, this second type of video chip uses a simplified system for displaying character-only graphics. In such a display, each character position corresponds to a byte in memory. Each byte merely holds a character code that is used by the video chip to access a ROM table of the actual eight-by-eight pixel character definitions. Thus a 40 by 25 character display requires less than one Kbyte. Colour implementations of this kind of system use another 1,000-byte area to hold colour data; each byte holds a colour code for the corresponding character position.

## VIDEO DISPLAY

The video display system used by the BBC Micro (and others) still has to use ROM character definitions, but these are accessed by the operating system when a character is to be written into the screen memory. The process of looking up the definition and performing the necessary manipulations to encode the integral colour information is done at the writing to screen memory stage, rather than at the memory-to-picture conversion stage.

Video displays based around this second method normally have a second, high-resolution mode in which the bits held in memory are in direct correspondence with pixels in the final display. This type of graphics display is known as *bit-mapping*. The colour information is not encoded within the display bits (as in the first system described) but is held in the same way as for the character-only display. That is, a 1,000-byte RAM area in which each byte holds colour information for an eight-by-eight pixel area of the screen.

On the Commodore 64, this allows high-resolution graphics, but text cannot be mixed in directly unless you write your own routines to write the actual character definitions into the bit map. Interestingly, this is the approach adopted by the Spectrum, which stores its screen as a simple bit map and uses a separate colour RAM to control each eight-by-eight pixel section of the screen. Characters and high-resolution graphics can be mixed on the Spectrum because the

operating system writes the actual character definitions into the bit map.

## SPRITES AND SCROLLING

The display of sprite graphics is controlled by the video chip (should the facility be present). Normally, sprites are defined by bit images in RAM and their positions on the screen, and other attributes such as colour and size, are controlled by internal registers on the chip. Because a single chip is used to handle normal graphics and sprites, video chips like the Commodore 64's VIC can be programmed to generate interrupts when sprites collide with other sprites or with background data.

The two types of video chip discussed here support scrolling in different ways. The second type performs scrolling through software. Since the number of bytes that make up the screen display is small, moving the screen data to new locations can be done quickly in machine code. The large number of bytes needed to hold the screen, in the first method, means that software scrolling is too slow. Video chips based on this system therefore incorporate a 'start of screen' register, which holds the address of the beginning of the screen memory. Scrolling can therefore be achieved by simply changing the address in this register, and by careful manipulation this method can be used to scroll the screen up, down, left or right.

The type of video chip used in the BBC Micro has much less work to do than, say, the VIC chip in the Commodore 64. It can therefore make its memory accesses during the phase of the clock cycle when the processor does not access memory itself. The video chip, therefore, does not slow the processor down. In addition to using the video chip, the BBC Micro incorporates a custom-designed ULA, which provides timing for the entire system, works out the relationships between logical and physical colours and provides RGB video output.

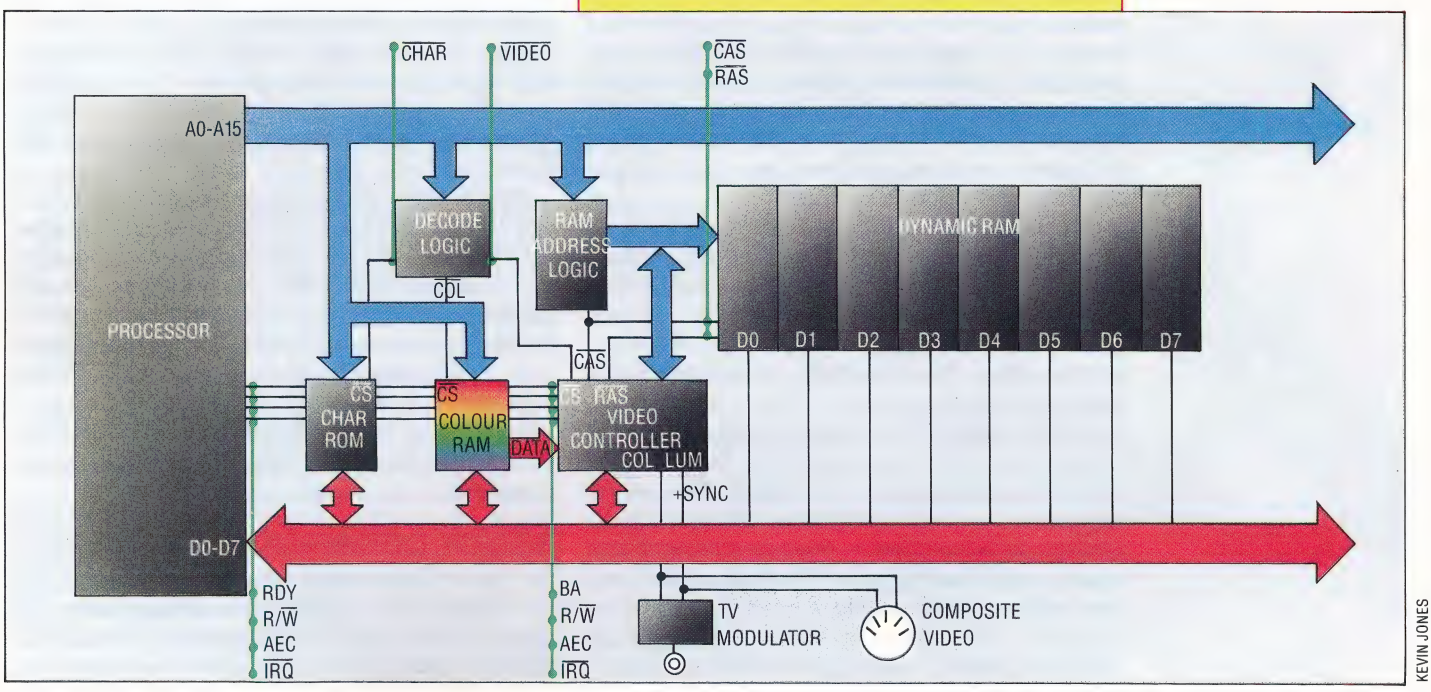
The VIC chip needs to make several accesses to memory. Some, such as RAM refresh and character data fetches (in the BBC Micro), are invisible to the processor, because they can be done on the alternate phase of the system clock. Some functions need data at a faster rate than this and processor operations need to be temporarily suspended while the video chip makes its memory accesses.

## Getting The Picture

The diagram shows how the Commodore 64's VIC chip connects to the system. VIC is connected directly to the data bus and uses the same multiplexed address lines as used by the dynamic RAM, which it also refreshes using the CAS and RAS control lines. Some of the video chip's operations can be performed without disrupting the processor. Also, the address bus enable control (AEC) ensures that the processor's address bus drivers are disconnected during the phase of the clock in which the video chip makes its accesses to memory.

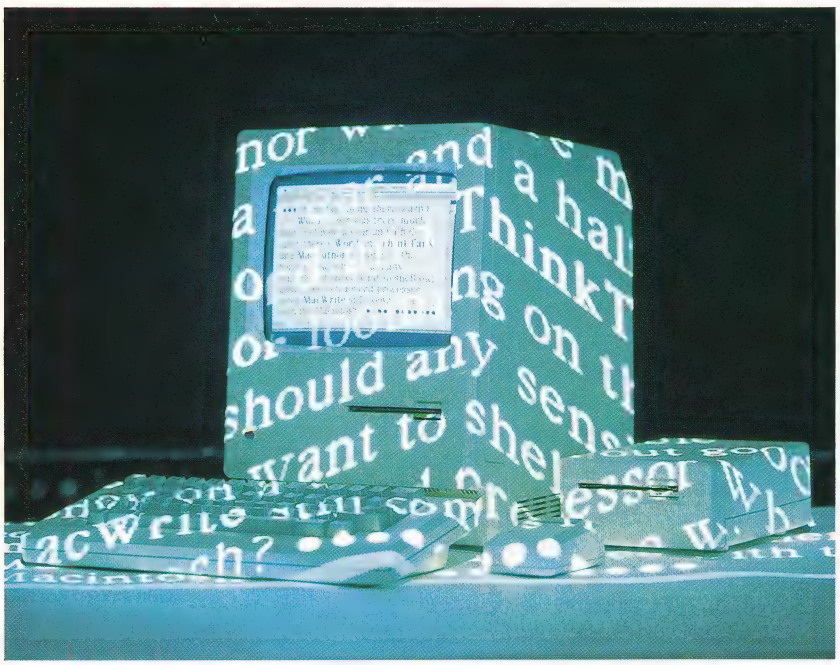
Accesses to the 1,000-byte display RAM and sprite data areas must be done faster than every alternate phase and so the processor needs to be disabled while the video chip 'steals' memory cycles to access these areas. This is achieved by linking VIC's bus available (BA) line to the READY pin on the processor. Three more clock cycles are allowed for the processor to complete any memory access operation. On the processor access phase of the fourth clock cycle, however, AEC will remain low to allow VIC access to memory during this phase.

It is obvious that maintaining the video display in this type of system slows the processor down, and in certain cases, where maximum processor speed is required, the screen should be blanked out




KEVIN JONES

# WORDS WITH A MOUSE



**Every Picture Tells A Story**
Like all packages written for the Macintosh, MacWrite makes extensive use of icons to execute commands positioned around the periphery of the screen. Many of these — such as the scroll bar — are commonly used on a number of Macintosh programs, whereas others are unique to MacWrite

**It's Written All Over It**
Apple has met a great deal of resistance to the idea of the Macintosh as a serious business machine. Part of the problem has undoubtedly been the lack of a comprehensive word processing package, which is a prerequisite for business use. However, a number of word processing programs have appeared, although they are only a handful compared with the hundreds that are available for the IBM PC

**The Apple Macintosh supports a wide variety of graphics capabilities under its WIMP environment, but it also accommodates a very efficient word processing system. MacWrite's methods of manipulating text are handled entirely by mouse control, doing away with the usual control characters.**

In contrast to the packages we've looked at in the previous instalments of this series, which are based on menus and control characters, MacWrite, the word processing package bundled with the Macintosh, relies heavily on the mouse-based pull-down windows and icons now expected from a Macintosh program.

On booting MacWrite, you are presented with the opening screen, consisting of a large window on which the document will be displayed. Above that is a series of icons and a ruler, and at the top is a menu bar consisting of six titles. Below this is the title bar where the name of the current document is shown.

Unlike most other packages, on which the cursor is moved with the aid of control keys, MacWrite manipulates it with the mouse controller. On other word processors, the cursor is used to set a position in the text that indicates where a particular option is to be performed. It can be positioned, for example, to insert text or create a block. The cursor is used identically in MacWrite, but it is also used to move the icons, which alter the Tab and margin controls, as well as pull down and select items on the menus.

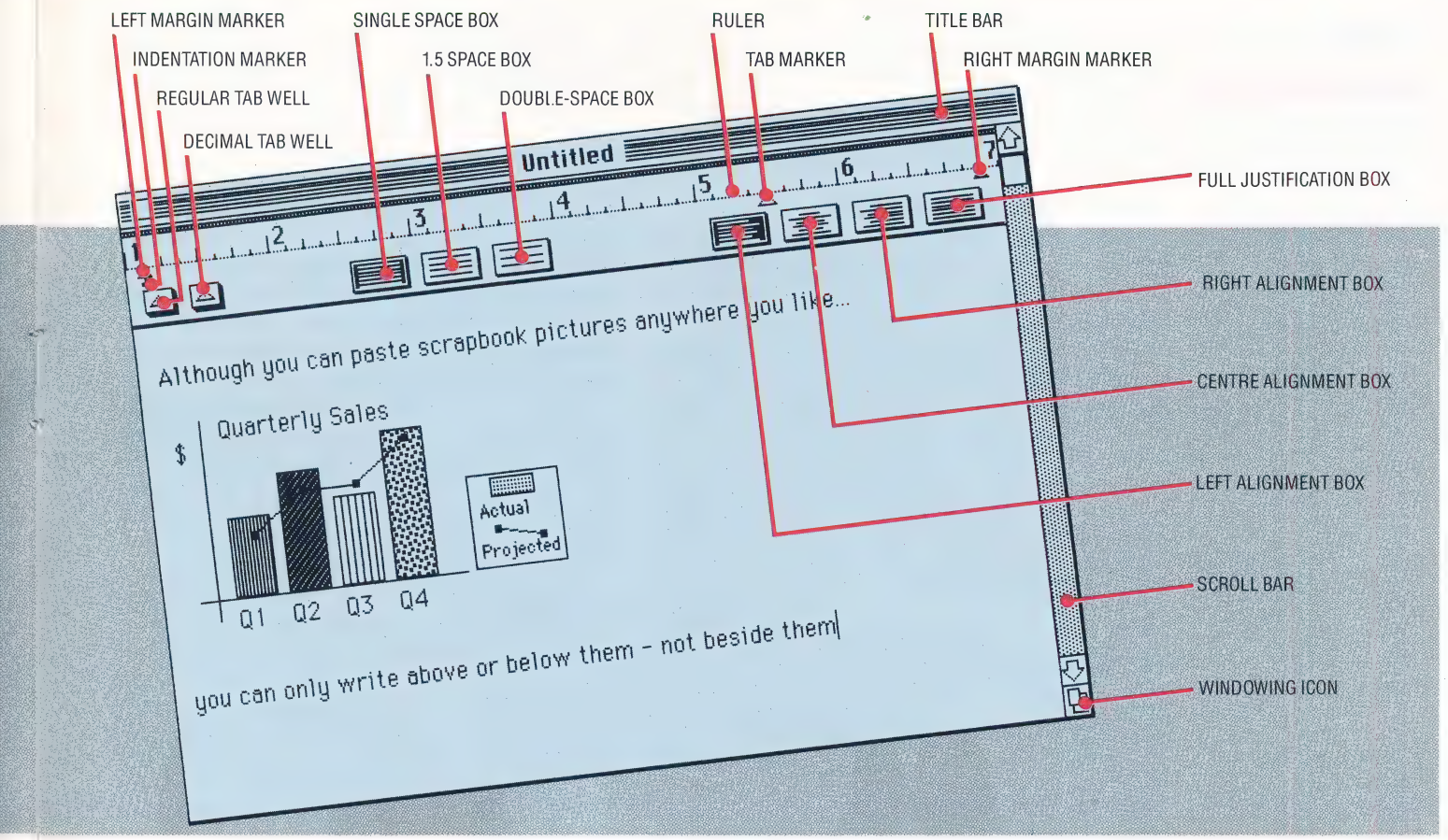
Pulling down a menu displays a number of different options, but not all of them are immediately usable. Those available are shown in black print while the others are shaded grey. The menu will often consist of a number of preset options, with the currently selected one having a tick next to it. Selecting another preset option will move the tick to that selection, indicating that this is the current default option.

The File menu contains the DOS commands that allow you to open and close files, as well as save them under the current, or a different, filename Also included in this menu is the Print command.

The second of the menus listed is Edit, which provides a number of commands for manipulating the text. Perhaps the most important of these for the beginner is the Undo command, which will erase all of the actions performed since the mouse button was last pressed. This might seem drastic, but many errors will occur due to incorrect selections with the mouse. Thus very little valuable work will actually be lost using Undo. Also included in this menu are the block movement commands.

## BLOCK COMMANDS

As with WordStar, blocks are created by delimiting the required text. After placing the cursor at either the beginning or end of the block and pressing the mouse, the cursor will change to




an icon known as the 'insertion point', which will allow you to insert text from that position. If you move the cursor up or down from this point, however, the corresponding text will be shown in inverse video, which in effect marks the block. A second press on the mouse will then set the other end of the block.

Once marked, a number of different operations can be performed on the block. By returning to the Edit menu, for example, you can select the Cut option, which places the block in a 'clipboard'. This is a buffer that stores the block before operations are performed on it. To view the contents of this buffer, you select the Show Clipboard option in the Edit menu. By setting another insert point and returning to the Edit menu, you can select the Copy or Paste options, which allow you either to copy or transfer blocks into that position. Note that any blocks involved in these operations will automatically be reformed.

The Search menu contains the Find/Replace commands and is identical to that implemented on most other word processing software.

## A RANGE OF FOUNTS

Many word processors allow different founts to be used within a particular document, but most of these are limited to those used for emphasising, such as italics and bold face. Furthermore, because most packages confine themselves to using the character matrices already held within the computer's operating system ROM, it's usually impossible to see exactly how the document will appear until it has been printed.

MacWrite, however, contains many different founts, with more being added as each new version of the software appears. Also, because the Macintosh text screen is bit-mapped, rather than having the usual character cell arrangement, these founts can actually be displayed on screen. Despite this facility, however, the system is not perfect. Many of the founts, such as London (a Gothic style) are almost illegible on screen.

This bit-mapping facility is used to somewhat better effect within the Style menu, a series of options providing the more usual forms of emphasis and effect, such as underline and bold face print. These can be printed on screen in order to gauge their effect before sending the document to the printer.

Most of MacWrite's page layout functions are held under the Format menu, although the actual page size formats are held in the File menu. Headers and footers (standard titles that will appear on the top and bottom of each printed page) can be set, as well as the page number, time and date. The last three are displayed as icons that can be 'dragged' from their own menu bar and placed anywhere in the header or footer area.

Underneath the ruler at the top of the screen is a series of small arrow icons that can be dragged along to set the margins, paragraph indentations and Tab settings. The document will be reset automatically when any parameters are changed. Two boxes below and to the right of the ruler are 'Tab wells', from which you can obtain further

**MACWRITE**

**Wordwrap**

★★★★★

Wordwrap and automatic justification are fully implemented in MacWrite.

**Block Movement**

★★★★★

Text and graphics can be moved and positioned anywhere within a MacWrite document.

**On-Screen Help**

Not implemented.

**80-Column Screen**

★★★

MacWrite supports only a 60-column screen in default mode, although 9 point text (the smallest print size) will allow 80 columns.

**Word Count**

No word count facility is provided in MacWrite.

**Find/Replace**

★★★

MacWrite enables up to 44 characters to be found and altered in the current window.

**WYSIWYG**

★★★★★

The ability to show various founts and diagrams on-screen is one of the major features of MacWrite.

**Mailshot Facility**

Not implemented.

**Spelling Checker**

Not implemented, although utilities from third party manufacturers are available.

**Founts Available**

★★★★★

MacWrite implements a wide variety of founts, although the actual number varies between different versions.

**File Linking**

★

Although some features — such as Cut and Paste — can be transported between documents, there is no file linking facility as such.

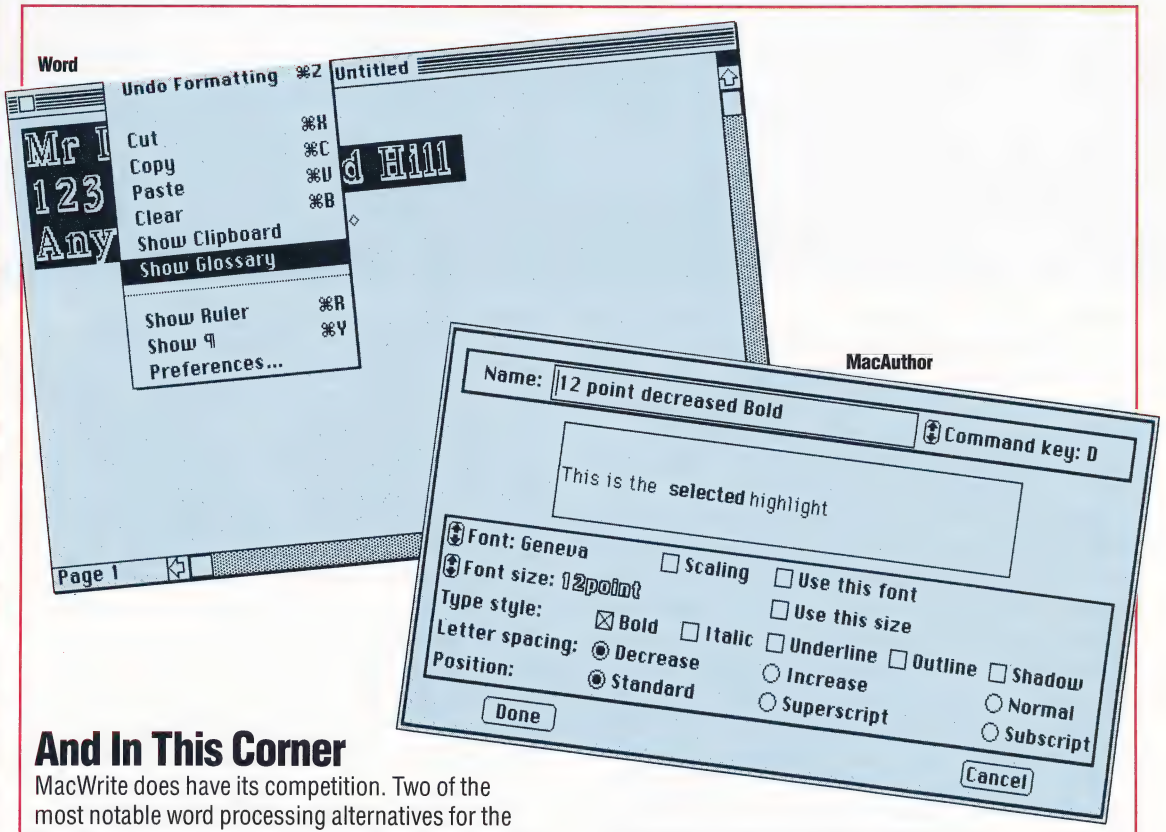
icons and set them on the ruler. To the right of these are a number of page icons, each having lines to indicate their functions. The first three correspond to single, one-and-a-half and double line spacing, while the remaining four at the far end of the bar indicate justification.

## TEXT AND GRAPHICS

The Macintosh allows text and graphics to be combined on the page and displayed on screen (see page 1581). MacWrite supports this facility, although pictures and images have to be drawn with MacPaint and MacDraw, the art packages bundled with the Macintosh. Pictures can be accessed by means of the Scrapbook facility, which is part of the Macintosh's built-in desktop. The Scrapbook is an area of memory set aside to store and transfer both text and graphics anywhere within a document. Furthermore, because of the machine's flexibility, the pictures can be adjusted to any size required.

Although MacWrite is an extremely easy package to learn and use, it nevertheless has its drawbacks. To begin with, the program leaves little memory space available for text on the standard Macintosh. This might be difficult to believe considering the machine has 128 Kbytes, but the operating system and bit-mapped screen use up a lot of space. Also, the fact that MacWrite's text resides entirely in the computer's memory (as opposed to some other advanced word processors that continually save the text to disk) means that only about five pages or so can be written into the machine before it runs out of memory.

For all of that, though, MacWrite is an unusual package based around an unusual operating system. The way in which the WIMP format has been adapted for an application that has previously relied entirely on keyboard entry is both intriguing and ingenious.




## And In This Corner

MacWrite does have its competition. Two of the most notable word processing alternatives for the Macintosh are Word from Microsoft and MacAuthor from Icon Technology. Both address the central problem with MacWrite — it is unable to handle more than a few pages of text. At first glance, both packages appear similar to MacWrite, with icon-driven commands and a screen surrounded by the scroll and title bars. However, they contain additional features that make the Macintosh a much more serious proposition for attempting professional word processing.

Word contains a number of features intended to allow greater flexibility and power, enabling up to four documents to be manipulated on screen at once. It also offers an improved type of Glossary, rather than relying in the Scrapbook facility, which has its limitations.

While Word is intended as a business word processing system for the Macintosh, MacAuthor, as the name suggests, is aimed at writers. It has been developed to enable the greatest possible flexibility in the use of characters. The user can create new characters, for example, by combining those within the existing character set and altering the 'style' of the text at will. Because of this greater flexibility, MacAuthor is somewhat harder to get to grips with than Word or MacWrite. Despite this, many professional writers may find the extra effort worthwhile

# STRUCTURAL SURVEY

**We continue our look at C by examining some of the control structures available in this language. Beginning with the logical operators and moving on to loop structures, we'll see some striking similarities to those found in BASIC and PASCAL.**

The ordinary relational operators in C are the same as for most other languages — namely <, <= — >, >= -, but note that the symbol for equality is == and for non-equality is !=. The logical connectives are && for AND and || for OR. Operator precedence is as usual, so that complicated logical expressions can be constructed in much the same way as in BASIC. These expressions assume greater significance when you remember that an assignment has a value and can therefore be tested in a logical condition. By using the standard library function getchar(), for example, which fetches the next character from the standard input device, we can construct a condition:

```
char_count (max_line_length – 1 && (c=
    getchar()) != /n' && c != EOF
```

This will first test for a maximum number of characters and then, if this has not been reached, fetch the next character and check to see if it is an end of line or an end of file, all in one logical expression.

A further feature is that logical expressions also have values; false is zero and true is one (in fact, any non-zero value will be taken as true). So, if necessary, logical expressions can be used in calculations. Conversely, simple numeric values can be tested directly without using any operators. The unary negation operator ! can be placed in front of any integer value and will change it to zero if it is non-zero, or to one if it is zero. Thus, the test:

```
if (!int_value)
```

is equivalent to:

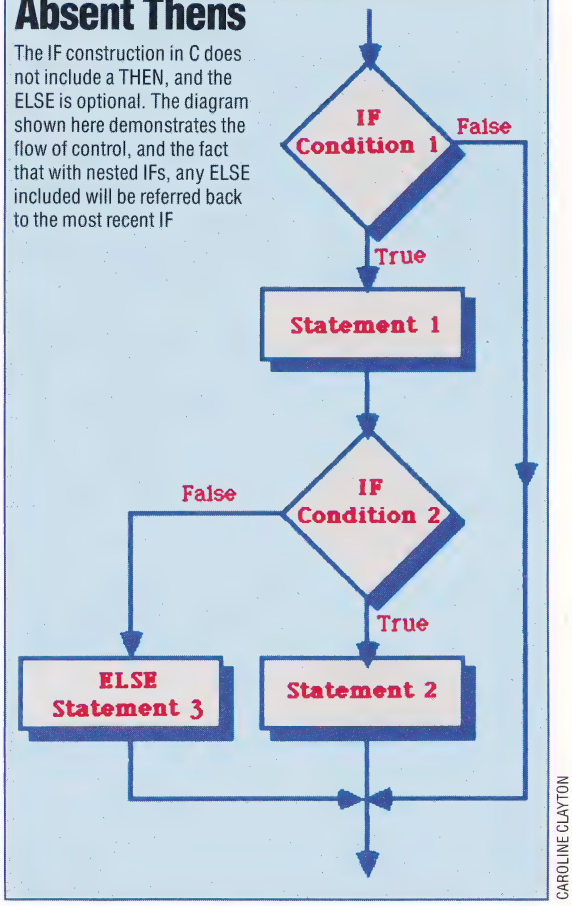
```
if (int_value == 0)
```

In addition to the normal set of logical operators, C has a number of bitwise logical operators that can be used on any integer or char types to provide bit manipulation as provided by most assemblers. These are:

| | |
|---|---|
| & | bitwise AND |
| \| | bitwise inclusive OR |
| ^ | bitwise exclusive OR |
| <<n | where n is an integer, shifts left n bits |
| >>n | where n is an integer, shifts right n bits |
| ~ | one's complement |

## Absent Thens

The IF construction in C does not include a THEN, and the ELSE is optional. The diagram shown here demonstrates the flow of control, and the fact that with nested IFs, any ELSE included will be referred back to the most recent IF




CAROLINE CLAYTON

As an example, the & operator is often used to mask off certain bits so that one or more in a word can be tested. To check if bit 3 is set to one in a single byte, c, we can use:

```
if (c & 0x08)
```

Note the notation 0x, which precedes a hex constant — a number that begins with a zero is taken to be octal.

Selection in C is taken care of by the usual sort of IF statement, which takes the form:

```
if (expression)
    statement_1
else
    statement_2
```

where the expression is normally a logical expression, but as we have already seen, it can also be any expression that gives an integer value. Note that there is no then and, as usual, the else part is optional.

The statements can be either simple or else compound statements enclosed in braces, { }. Nested ifs can be a problem as with many

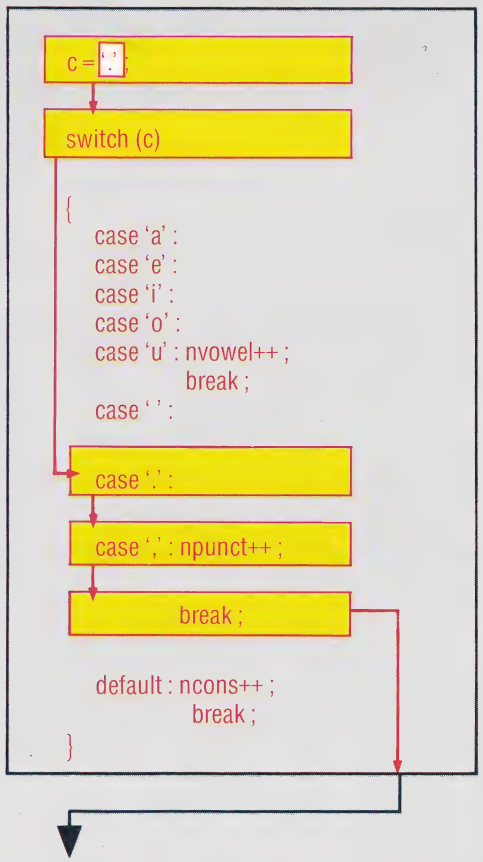
languages, but the usual rule is applied so that each else is matched with the most recent if. Should you want to change this, braces can be used.

More complex selections, from more than two alternatives, can be handled by the switch construct, which performs the same sort of function as the PASCAL case statement. The format of switch is:

```
switch(integer_expression)
{
case value_1 : statement_1 ;
case value_2 : statement_2 ;
.....
default : default_statement ;
}
```

The case values must be constants and all different; the default case is optional. The case values serve as labels so execution does not automatically transfer to the end of the case statement when one of the cases has been dealt with. Instead, it 'falls through' to the next case.

To prevent this, the break statement can be used to transfer control to the statement following the switch. An example of this is shown.




## Switching Control

The SWITCH construct acts in similar way to an ON . . . GOTO construct in BASIC. The parameter of SWITCH is evaluated and control passed to the relevant CASE. The example shown here adds 1 to the appropriate count variable (nvowel, npunct, or ncons) according to the value of the variable c. This provides a means of counting the number of occurrences of vowels, consonants, and punctuation symbols. The variable c would normally be an input value, but here, to show the flow of control in a particular instance, it has been assigned the value '.'. Note that the 'fall through' from one CASE to the next provides a convenient way of providing for multiple cases with the same action. The final 'break' is not strictly necessary, but is good practice just in case you decide to include an extra CASE at the end at some later date

## LOOPING STRUCTURES

Iteration, or 'looping', can be handled in three ways. The most commonly used construct is the while loop, which simply takes the form:

```
while (condition) statement ;
```

As usual, the statement can be simple or compound (in braces).

It is also common, if not necessarily good practice, to put as much of the body of the loop in the condition, remembering that the condition can be any integer expression. The following examples represent two ways of writing the same loop to input a sequence of digits as characters and convert them to an integer number, ending with the first non-digit:

```
int num ;
char c ;
num = 0 ;
c = getchar() ;
while (c >= '0' && c <= '9')
{
    num = num * 10 + c - '0' ;
    c = getchar() ;
}
```

Note the way in which the characters c and '0' can be used in arithmetic. The ASCII code is automatically taken in these cases and the values converted to type int for the calculation. The second method puts the assignment to c in the loop condition:

```
num = 0 ;
while ((c = getchar()) >= '0' && c <= '9')
    num = num * 10 + c - '0' ;
```

Note here that there is only a single basic statement to be repeated so that there is no need to use braces.

The second way of looping in C is to use the for statement. This is similar in some respects to the BASIC FOR...NEXT loop, in that there is an initialisation, a terminating (or continuing) condition and a 'step function' that gets repeated every time round the loop. In BASIC, as in most other languages, the initialisation is an integer value in an integer variable, the terminating condition is a final value that this variable must reach, and the step function simply adds on an integer value. In C, this statement is much more general:

```
for(exp1 ; exp2 ; exp3)
    statement ;
```

where exp1 represents initialisation and is carried out before the statement is first executed. The continuation condition, exp2, and the loop will be repeated as long as this expression remains true (or non-zero); exp3 is carried out at the end of each repetition of the statement. The BASIC:

```
FOR I=1 TO N
statement (s)
NEXT I
```

has as its equivalent in C:

```
for(i=1 ; i <=n ; i++)
statement;
```

To illustrate the flexibility of the for construction and the power of the language, we could code the same loop as in the WHILE loop example to convert an input string of characters to an integer number as follows:

```
for(num=0 ; (c=getchar()) >='0' && c <='9' ;
    num=num * 10+c-'0') ;
```

Notice that the entire body of the loop is incorporated into the condition. This is often possible in C but it is debatable whether or not it's good practice. The code that is produced is certainly compact and efficient, but it is also very cryptic.

There is a third way of looping in C, which is the equivalent of a PASCAL repeat . . . until loop. Unlike the other two constructs, the testing is done at the end of the loop rather than at the beginning, and takes the form:

```
do
    statement
while(condition) ;
```

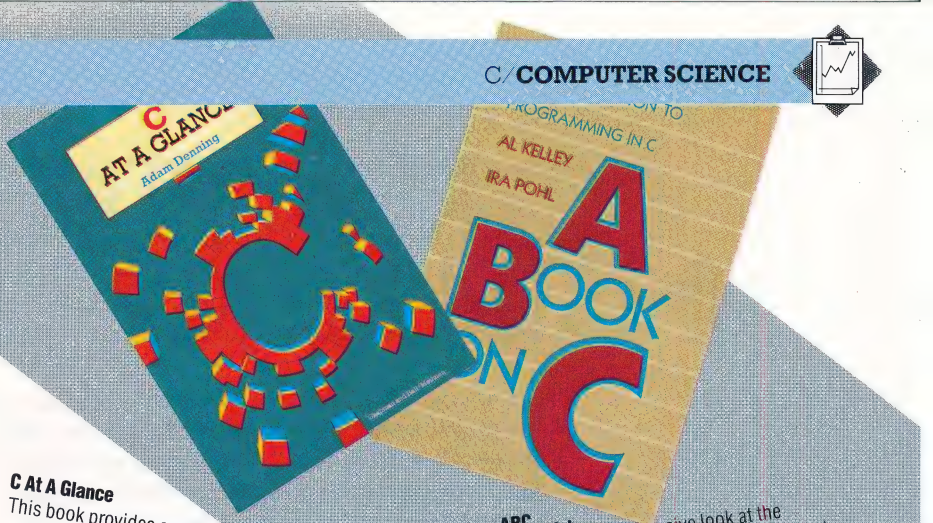
This construct, however, is very rarely used.

## BREAK AND CONTINUE

There are two additional statements that make loops rather easier to use than in other languages. We have already come across the break statement. If a break is encountered in the middle of a loop, execution will immediately transfer to the statement following the end of the loop. The following example will read a succession of characters terminating with a Return, but it will also break out of the loop if a Control C (ASCII code 3) is typed:

```
while((c=getchar()) !='\n')
{
    if c==3
        break
    else
        /* continue processing c */;
}
```

The statement works in a similiar way but simply moves on to the next execution of the loop rather than breaking out entirely. It is not very commonly used.




**C At A Glance**
This book provides a rather more in-depth treatment of its subject than the title would suggest. The book assumes a knowledge of BASIC and approaches C from that standpoint, making it readily accessible to most home computer programmers. Published by Chapman and Hall and written by Adam Denning, it retails at £7.95 (ISBN 0-412-27140-0)

**Learn Your ABC . . .**
A Book On C takes a comprehensive look at the language and explores associated subjects — Unix in particular with which C is closely associated. Written by Ira Pohl and Al Kelley, it costs approximately £15 (ISBN 0-8053-6860-4)

The existence of break and continue removes nearly all need for a goto statement. There is a goto in C, together with a mechanism for labelling statements, but it is never necessary to use it and very bad practice to do so. Beyond acknowledging its existence, we'll not mention it any further. If you really must write a C program using gotos any comprehensive text book on C will suffice.

## Prime Imperative

```
/* Program to print all prime numbers less than
1000 */
main ()
{
    int prime_count = 0, limit = 1000, divisor,
    number_to_test ;
/* Note the initialisation of the two variables, prime_
count and limit, in the declaration statement */
    for (number_to_test = 2, number_to_test <
    limit, ++number_to_test)
/* main loop to test all numbers between 2 and 1000
*/
    {
        divisor = 1 ;
/* look for divisors using % (mod) operator */
    while (number_to_test % ++divisor != 0) ;
/* incrementation is carried out before the test */
/* number is prime if divisor reaches the value of
number_to_test */
    if (divisor == number_to_test)
/* add 1 to count */
    {
        ++prime_count ;
/* print prime number, ten on a line */
            printf("%6d", number_to_test) ;
            if (prime_count % 10 == 0)
                printf("/n") ;
    }
}
printf("\n\nthere are %d prime numbers less
than %d\n",
    prime_count, limit) ;
}
```

**Prime Example**
Our sample program counts and prints all the prime numbers between 2 and 1000. Note the compact code produced using the pre-incrementing instruction ++ and the format of the FOR loop

# READ THE INSTRUCTIONS

**We begin a detailed look at the instruction set of the Motorola 68000. Here, we discuss the simplest examples: data copying instructions such as MOVE and PEA, and several of the arithmetic instructions.**

In the previous two instalments, we examined the way the 68000 addresses its operands, and showed you how we use some of the basic instructions (such as MOVE, ADD and the generalisation 'OPCODE'). Let's now move on to look at the instruction set in a bit more detail, starting with the data copying instructions and then moving on to the computational instructions, which involve binary arithmetic.

We will demonstrate the usefulness of more important instructions, and highlight any significant detail or possible pitfall in using them. If you intend to program the 68000 extensively, however, you should consider purchasing either the *Motorola 68000 User Manual* or one of the other publications mentioned in this series.

Before considering the instructions, we need to examine in a little detail the contents of the status register (SR). Half of this register contains the condition codes that indicate the result of the last executed instruction. Each condition code may be considered as a one-bit memory allocated to a particular arithmetic condition. Let's examine each of these in turn:

• *BIT 0: Carry Bit or C Bit:* This bit is set when an arithmetic operation produces a carry out from the most significant bit of the data operand. For example:

```
adding 01100000
with    11100000
gives 1 01000000
```

In this case, the 1 is carried out from the most significant bit of the sum, which is of byte-length.

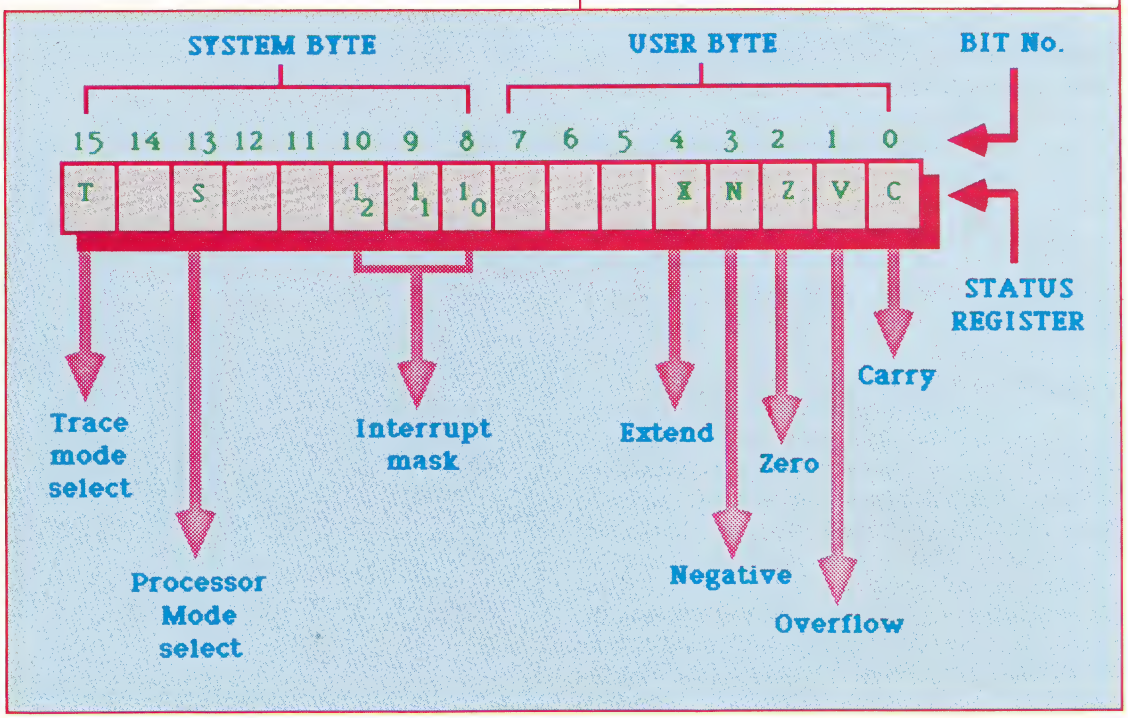
In this example the carry out is moved into the C bit of the SR, and not into the least significant bits of the next data size up (a word in this case). Also note that the carry out may or may not be significant — it depends entirely on what you are doing. If, for example, you were computing some multiple precision result (over more units of the data operand) then the C bit should be significant.

• *BIT 1: Overflow Bit or V Bit:* This bit is set when the result of the computation will not fit in the bit range of the data operands. For example, adding 1

## Exalted Status

The 32-bit status register is divided into system and user sections, the bit significance of which is shown here. Note the status bits used to flag the processor modes — the 68000 can operate in two modes, supervisor and user. The reason for this is that in a multi-processing environment it is necessary to be able to keep an eye on the different tasks being executed. Typically, tasks will be carried out by the 68000 in user mode (which has its own separate stack pointer and does not allow alteration of the system status bits), and system software (such as for overseeing tasks) is executed in supervisor mode.

To begin, you will generally find it more convenient to use the 68000 in supervisor mode, in which all opcodes are legal, rather than dealing with the problem of avoiding privileged instructions in user mode. Additionally, the 68000 possesses a 'trace' mode — this is a powerful debugging facility that allows the user to 'single-step' through instructions, while the 68000 automatically calls a user debug routine after each instruction



to 32767 (the maximum positive integer for a 16-bit word) will give overflow for word data operands, and the binary result will be meaningless.

- *BIT 2: Zero Bit or Z Bit:* This is set when the result of a previous computation is zero.
- *BIT 3: Negative Bit or N Bit:* This is set for negative results.
- *BIT 4: Extend Bit or X Bit:* This bit is used for multiprecision operations, but it is generally the same as the C bit (although not affected by MOVE instructions).

## DATA COPYING INSTRUCTIONS

The 68000's data copying instruction is MOVE, which copies from the source to the destination. It is a versatile instruction — any source addressing mode can be used with it and most addressing modes can be used as destinations (with the exception of address registers, PC relative modes and immediate mode). This group of addressing modes is known as the *data-alterable mode*, and there is also a subset of this grouping called the *memory-alterable modes* (data-alterable less the data registers). Both of these groupings will be discussed later.

Returning to the question of using the MOVE instruction — neither of the following move instructions is allowed:

```
RORG  $1000
MOVE  D2,BETTY  PC relative not allowed for BETTY
MOVE  D2,A2     Address register not allowed as
                destination
```

Note that only the N and Z bits in the SR are affected by the MOVE instruction, V and C being set to zero.

To get over the problem of address registers as destination operands, there are two options:

- Using MOVEA, which takes the contents of the source operand and copies it to the destination address register.
- Using LEA, which takes the source address (usually absolute) and copies this into the address register.

Neither of these two instructions will affect the condition codes. In the same way, there are special instructions to move data to and from the status register and the user stack pointer, but these are more concerned with systems programming.

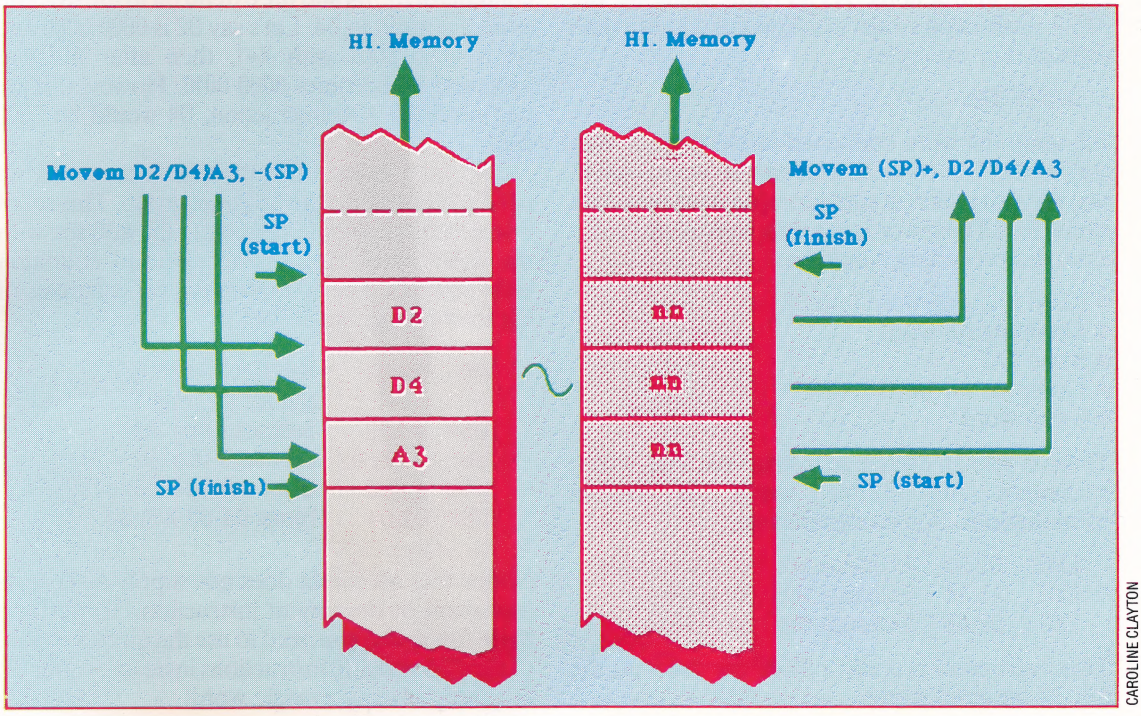
Another extremely powerful data copying instruction is MOVEM, which allows you to save or retrieve any declared registers (address or data) to or from consecutive memory locations. This means, for example, that on entry to a subroutine any registers that would otherwise be corrupted in the subroutine can be saved, and then restored on exit. For example:

```
entry  MOVEM   D2/D4/A3,PAD
                (subroutine code uses D2, D4
                and A3)
exit   MOVEM   PAD,D2/D4/A3
```

This will save D2, D4 and A3 in PAD on entry, and then restore these registers on exit. Alternatively, the registers could be saved on the stack. So, for example, we could have:

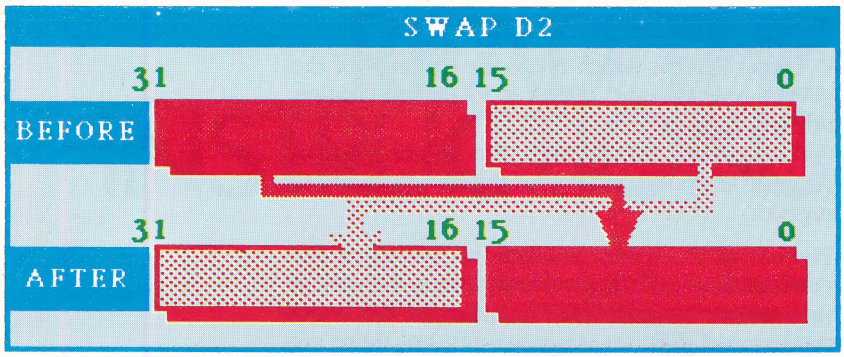
```
entry  MOVEM   D2/D4/A3,-(SP)
                subroutine code
exit   MOVEM   (SP)+,D2/D4/A3
```

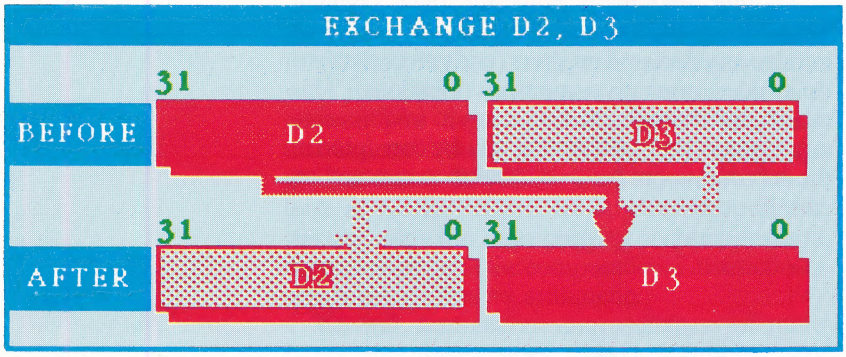
Another variation on the MOVE instruction is the quick MOVE (or MOVEQ) instruction. This is useful when setting up signed eight-bit constants (+127 to -128) in a data register within one memory word. Typical uses would be setting up loop counts




CAROLINE CLAYTON

**Move Along, Please**
The powerful MOVEM instruction lets you manipulate 'register lists' in sequence within one instruction. Its use is shown here in conjunction with the pre-decrement and post-increment instructions to push and pull three registers onto the stack. The position of the stack pointer is shown before and after each instruction




**Want To Make A Swap?**
Using long word storage may prove rather unwieldy where word- or byte-length operands are required. SWAP helps us get round this by transferring the values in bits 0 to 15 of a register with those in bits 16 to 31. Additionally, EXG (the exchange instruction) enables the swapping of long words, as shown in the lower diagram

into a data register. For example:

```
MOVEQ #34,D2
```

would set up 34 in D2 in one memory word. Note that if you leave off the Q, some assemblers do not assume the quick mode. Therefore, the instruction MOVE #34,D2 would be coded into two words.

Another data copying instruction which can be useful for stack operations is PEA, or 'push effective address' onto the stack. For example, PEA BETTY will push the address of BETTY onto the stack — perform -(SP).

Finally, let's have a look at the register swap and exchange instructions. As its name suggests, the SWAP instruction swaps whole words within a data register. For example:

```
SWAP  D2
```

will swap the word contained in bits 0 to 15 with the word in bits 16 to 31. This can be very useful if, say, we wish to repeat some computation on words stored in full long words in a data register. In such a case, the procedure would be:

- Load full long word into D2
- Compute on word
- Swap D2
- Compute on word
- Swap word

There is also an exchange instruction, EXG, which exchanges full 32-bit words in certain combinations. Some examples are:

```
EXG   D2,D3   Exchange between data registers
EXG   A3,A4   Exchange address registers
EXG   D2,A5   Exchange data and address
```

This instruction is really a SWAP for use with the full 32-bit sized words.

## INTEGER ARITHMETIC

These instructions form the basis of all arithmetic computations (whether on fractions, real or double-precision objects) that involve, at the basic level, adding binary bits together. Even if your application does not involve numerical computation, you still need to be able to carry out simple arithmetic operations so that you can, for example, convert character codes or form array indexes.

The ADD instruction simply adds the source to the destination and stores the result in the destination. The data objects can be any of the data attribute sizes, and all the condition codes are affected. For example, ADD.W D2,D3 will add the word contents of D2 to D3 and store the result in D3.

All the addressing modes can be used for the source data, but you need to use a special instruction, ADDA, if your destination is an address register. So, ADDA D2,A4 will add the contents of D2 to A4.

In the case of immediate data, there is also a special instruction, ADDI. This adds the immediate data (all attributes allowed) stored as an extension word to the destination (only data-alterable modes allowed). So, ADDI #3423, BETTY will add 3423 to the contents of BETTY.

As with the MOVE instruction, there is also a quick form of ADDI called ADDQ. Using this instruction, however, the data is restricted to the range 1 to 8. Thus ADDQ 5,D2 would add 5 to the contents of D2, and the whole instruction occupies one word.

An important point to note with the ADD instructions we've looked at so far, is that any carry will not be included in the destination sum. If we want to do this — particularly with multiple-precision arithmetic — then we need to use the ADDX instruction. ADDX D2,D4 will add up the contents of D2, D4 and the extend bit in the SR, and store the result in D4. Let's say D2 is 0000 0000 and D4 is 0000 0001 with X=1, then after ADDX is executed D4 becomes 0000 0010. However, if we had used the ADD instruction, D4 would be 0000 0001.

The next group of arithmetic instructions we need to consider is the SUBtract group. This set of instructions is complementary to the ADD set, with the same addressing restrictions and condition code settings. The following typical examples of the instructions are all valid:

```
SUB   D2,D3    Subtract D2 from D3
               and put the result in D3
SUBA  #4,A3    Subtract 4 from A3
SUBI  #200,D2  Subtract 2 from D2
SUBQ  #1,D2    Quick subtract 1 from D2
SUBX  D2,D4    Forms D4-D2-X in D4
```

Notice that Motorola does not supply a separate increment or decrement instruction. To perform these functions, you need to use the quick versions of the ADD and SUB instructions instead — after all, they only occupy a single word!

# HAVE YOU ORDERED YOUR VOLUME EIGHT BINDER YET?

FOLD 2

IF YOU HAVE NOT ASKED FOR THE BINDERS TO BE SENT TO YOU AUTOMATICALLY, DO SO NOW, AND ENSURE THAT YOUR COPIES OF THE HOME COMPUTER ADVANCED COURSE ARE KEPT PROPERLY BOUND AND IN GOOD CONDITION FOR YEARS TO COME.

BY TICKING THE BOX OPPOSITE, YOU WILL BE SENT BINDER NUMBER 8.

☐ PLEASE SEND ME MY VOLUME 8 BINDER NOW. I ENCLOSE A CHEQUE/POSTAL ORDER FOR £3.95 (WHICH INCLUDES POSTAGE AND PACKING).

FOLD 3

BY TICKING THE OTHER BOX *AS WELL*, YOU WILL BE SENT SUBSEQUENT BINDERS FOR YOUR COLLECTION.

☐ I WOULD ALSO LIKE TO RECEIVE FUTURE BINDERS AS THEY ARE ISSUED. I UNDERSTAND THAT I WILL RECEIVE A PAYMENT ADVICE FOR £3.95 (WHICH INCLUDES POSTAGE AND PACKING) WITH EACH BINDER. IF I AM NOT SATISFIED WITH THE BINDER, I CAN RETURN IT TO YOU, WITHIN 14 DAYS, AND OWE NOTHING.

*NO STAMP NECESSARY.*

JUST FOLD UP THE PAGE AS INDICATED, REMEMBERING TO ENCLOSE YOUR CHEQUE/POSTAL ORDER MADE PAYABLE TO ORBIS PUBLISHING, AND SEND TO US TODAY.

FOLD 4

FOLD 4

I enclose a cheque/postal order made payable to: Orbis Publishing Ltd. for a total of £____ which I understand includes the cost of postage and packing.

NB: Please allow 28 days for the delivery of your binders.

When you have completed the order form fill in your name and address in the space provided.

Then cut along the dotted line to detach the page, enclose your cheque/postal order, and fold the page carefully – following the instructions to complete the reply paid envelope.

NO STAMP NECESSARY.

IF YOU ALREADY HAVE AN ACCOUNT NO. FOR YOUR BINDERS PLEASE FILL IT IN HERE.

Complete the section below with one letter or figure per space.

| | | | |
|---|---|---|---|
| MR | | INITIALS | SURNAME |
| MRS | | | |
| MISS | | | |

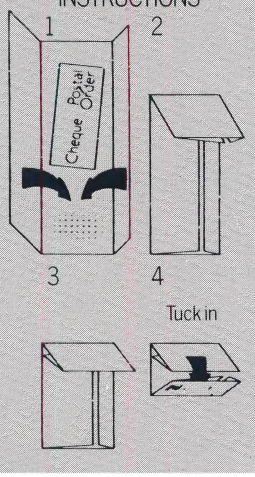
ADDRESS & POST CODE

TELEPHONE NO. INCLUDING STD CODE OR EXCHANGE NAME

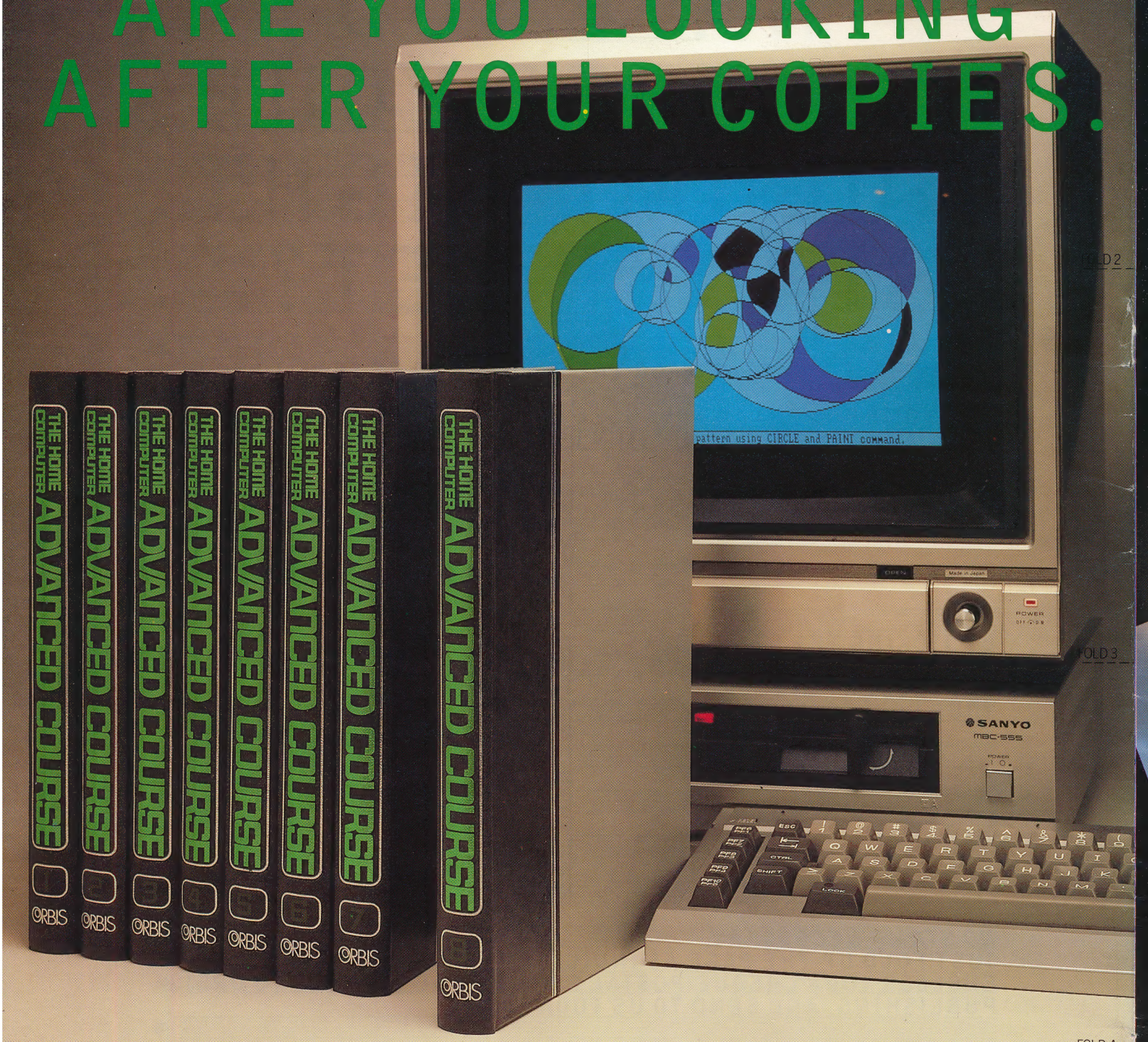
FOLD 1

FOLD 1

FOLD 4
FOLDING INSTRUCTIONS



# ARE YOU LOOKING AFTER YOUR COPIES.

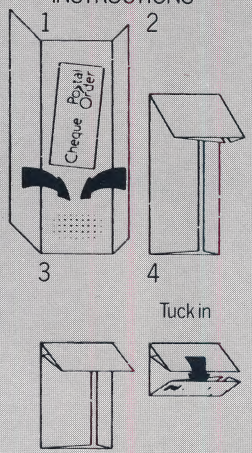



FOLD 2

FOLD 3

FOLD 4

FOLD 4
FOLDING INSTRUCTIONS




FOLD 1

Postage will be paid by licensee

Do not affix Postage Stamps if posted in Gt. Britain, Channel Islands or N. Ireland.

BUSINESS REPLY SERVICE
Licence No. SW4035.

The Home Computer
Advanced Course Binders
Orbis House, 20-22 Bedfordbury
London WC2N 4BR

2

FOLD 1

GUARANTEE GUARANTEE GUARANTEE GUARANTEE GUARANTEE

## GUARANTEE

**If you are not entirely satisfied with your binder, send it back immediately and it will be either exchanged, or, if you prefer, your money will be refunded in full.**

## GUARANTEE

GUARANTEE GUARANTEE GUARANTEE GUARANTEE GUARANTEE